CP/NET™

MICROCOMPUTER NETWORK CONTROL PROGRAM

USER'S GUIDE

DIGITAL RESEARCH™

CP/NET™

Network Control Program

Release 1.1

Release Notes

Digital Research
P.O. Box 579
160 Central Avenue
Pacific Grove, CA 93950
(408) 649-3896
TWX 910 360 5001

Dear CP/NET™ User:

Digital Research has introduced the CP/NET operating system as the next element in our family of portable operating systems. The development of CP/NET follows a progression of portable operating systems from Digital Research. It began with the single user CP/M® operating system. Next, the multiuser multiprogramming MP/M™ operating system was implemented to share the resources of the CPU and its peripherals. The CP/NET operating system was developed to share the peripherals and a common data base.

This shipment contains the 1.1 release of our CP/NET system. We have been pleased with the response from our CP/NET test site releases and we would like to receive your feedback regarding its design, possible extensions, and errors in implementation to maintain the same level of confidence that the computer industry has in our CP/M operating system.

There is no standard release configuration for CP/NET, although the distribution vehicle is configured for the Altos family of microcomputers. Equates and conditional assembly have been used in the sample SNIOS.ASM and NETWRKIF.ASM files to demonstrate the customization requirements of the portable release configuration of CP/NET.

Based on our experience and the experience of several CP/NET test sites, we estimate that 1 to 2 days are required to implement the portable version of CP/NET on computers that have running versions of CP/M 2.2 for the slaves, MP/M II for the masters, and are able to transmit and receive 8 bits of data between them. It can require 1 or 2 weeks to implement a highly optimized CP/NET system. The amount of time for such a reconfiguration will vary widely depending on the experience of the programmer and the complexity of the hardware on the target computers.

If you experience difficulties in reconfiguring CP/NET, contact the technical support staff, Digital Research, (408) 375-6262.

Technical Support Staff

Digital Research

CP/NET Network Control Program

Addendum

Compiled May 1982

Summary of Differences Between CP/NET[T.M.] 1.1 and CP/NET 1.0

1) CP/NOS[T.M.] is a product. The revision level is 1.1 to maintain compatibility with CP/NET.

2) Both CP/NET and CP/NOS are MP/M[T.M.] II compatible. CP/NET 1.1 should work under MP/M 1.1, but it is not supported, and we strongly recommend that all users still running under MP/M 1.1 upgrade to MP/M 2.0. Otherwise, the SLVSP process is compatible with the MP/M II SPOOL process, but not with the CP/NET 1.0 SPOOL&D process.

3) The CP/NET server maintains an internal FCB table for all file-oriented transactions. This prevents FCBs from being relocated on the requester without the server's knowledge. This table must be allocated in the NETWRKIF module.

4) Server FCB entries are 40 bytes long with the following format:

00-00	occupied field; flags table entries in use
01-02	address field; points to start of the FCB entry. Used as a checksum device to prevent inadvertent relocation of FCB table entries
03-03	open count field
04-39	FCB entry

(3) The requester's CCP.SYS records only after searching for a
Remote-File. This procedure can require at least twelve user
input (or system) exchanges between requester and server.

16) CP/M® 2.2 is patched in CCP.COM so all files after copying
to UP/M86 users.

17) CCP.C-E works properly with and without a Spooler. No
SPOOLIST is required to terminate listing.

18) Upon receipt of an end-of-file, the server decrements a
requester from a network printer if there is no spooler
in the server system.

19) The "Ms Fri-80" interface modules BIOS.A, MPMBIO.A, and CBASIC80
have been combined into a Printing Library, PRINT.LIB.
Existing PRI-80 printing CBASIC programs direct I/O interface
with this library by the program overlay up to 10K bytes
of memory. The old modules are still supplied in source
form, as are the external declaration modules.

20) MPMBIOS.ASM accepts a "wait-for-system-ID within failure.

CP/NET® Release 1.1
Patch 01, 5/06/82

Compiled May 1982

Products and Serial Numbers that require updating:

CP/M® V2.2 -- All serial numbers
CP/NET® V1.1 -- All serial numbers

Program: PIP.COM

Error Description:

Occasionally, the FCB table becomes full when using PIP with CP/NET because PIP does not close its input files. This patch forces PIP to close its input files.

Patch Procedure:

Make sure you have a back-up copy of PIP.COM before using DDT to make the following changes.

```
A> ren pip.sav=pip.com
A> ddt pip.sav
DDT VERS X.X
NEXT PC
1E00 0100
-a110                   ;make sure this patch is not
0110 lxi b,1e06         ;installed on top of drivers
0113 call 881           ;for reader and punch that are
0116 jmp  a97           ;normally installed in this area
0119
-a0a8e
0A8E jz 110
0A91
-g0


A> save 30 pip.com
```

Licensed users are granted the right to include these changes in CP/M and CP/NET V1.1 software.

CP/NET® Network Control Program Release 1.0
Application Note 01, 6/18/82

Compiled June 1982

LOADING THE CCP FROM OTHER THAN DRIVE A

This application note gives you the location of the File Control Block (FCB) for the CCP.SPR file, used by the NDOS to load the CP/NET® slave Console Command Processor (CCP).

The patch procedure shown below changes the drive from which the CCP.SPR file is loaded:

```
A>ddt ndos.spr

DDT VERS x.x
NEXT PC
0D80 0100
-s3af
03AF 01 02   ;02 = drive B:, set to any drive
03B0 43 .
-g0

A>save 13 ndos.spr
```

CP/NET™
Control Program for a Microcomputer Network
User's Guide

Digital Research
P.O. Box 579
160 Central Avenue
Pacific Grove, CA 93950
(408) 649-3896
TWX 910 360 5001

COPYRIGHT

DISCLAIMER

Digital Research makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Further, Digital Research reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of Digital Research to notify any person of such revision or changes.

TRADEMARKS

CP/M is a registered trademark of Digital Research. ASM, DDT, MAC, CP/NET, CP/NOS, MP/M, MP/NET, LINK, RMAC, and PL/I-80 are trademarks of Digital Research. Z80 is a registered trademark of Zilog, Inc. Intel is a registered trademark of Intel Corporation. PL/M is a trademark of Intel Corporation. DSC-2 is a trademark of Digital Microsystems. DB8/2 is a trademark of Dynabyte.

The CP/NET User's Guide was prepared using the Digital Research TEX Text Formatter and printed in the United States of America by Commercial Press/Monterey.

****
* Fourth Printing: May 1982 *
****

Foreword

The CP/NET™ User's Guide is a manual for several different levels of CP/NET users. Section 1 contains all the information for you to use the network when executing CP/M® application programs. You need no skill level beyond that required for normal CP/M operation.

Section 2 describes the CP/NET interprocessor message format and each of the Network Disk Operating System (NDOS) functions that can be invoked from application programs. This section provides the necessary information for you to access the network primitives.

Section 3 provides information for the systems programmer. This section describes how to write a custom Slave Network I/O System (SNIOS) that performs the CP/NET requester network functions. The mechanics of implementing and debugging a custom SNIOS are also discussed. Programmers attempting to develop an SNIOS should be very familiar with CP/M and have experience in writing a custom CP/M BIOS.

Section 4 provides the information for the systems programmer to write a custom Network Interface Process (NETWRKIF) that performs the CP/NET server network functions. This section also discusses the implementation and debugging of the NETWRKIF module. You must have a high degree of competence and experience with MP/M™ to develop a custom NETWRKIF. You must be familiar with the process and queue descriptor data structures and the MP/M XDOS primitive functions.

Note: throughout this guide, the terms slave and requester are synonymous, and the terms master and server are synonymous.

Table of Contents

Table of Contents
(continued)

Appendixes

Section 1
CP/NET Features and Facilities

CP/NET, a network operating system, enables microcomputers to access common resources via a network. CP/NET allows microcomputers to share and transfer disk files, to share printers and consoles, and to share programs and data bases. CP/NET consists of servers running MP/M and requesters running CP/M. The servers are hosts that manage the shared resources that the network requesters can access.

By separating the logical operating system from the hardware environment and placing all hardware independent code in a separate I/O module, CP/M and MP/M have gained widespread industry acceptance. CP/NET has this same design approach. CP/NET is network independent; all network dependent code for the requester has been placed in the Slave Network I/O System (SNIOS) module, and all network dependent code for the server has been placed in the Network Interface Process (NETWRKIF) module. Logical messages passed to and from the SNIOS or NETWRKIF are transmitted over an arbitrary network between servers and requesters using an arbitrary network protocol.

CP/NET is the first of a family of network operating system products from Digital Research. CP/NET is a bridge between a microcomputer running MP/M and a microcomputer running CP/M. The MP/M server manages resources that are considered public to the network. The CP/NET requesters executing CP/M have access to the public resources of the server and their own local private resources that cannot be accessed from the network. This architecture guarantees the security of the requester's resources while still permitting the server's resources to be shared among the requesters.

The MP/M server also responds to the network asynchronously in real-time, while the CP/M requesters perform sequential I/O and are not capable of monitoring a network interface in real-time. The following figure illustrates the relationship between CP/M, MP/M, and CP/NET.

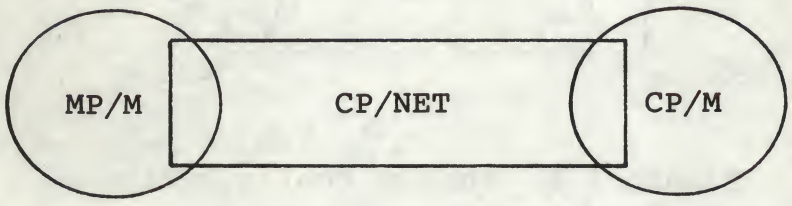


Figure 1-1. Standard CP/NET Configuration

The second network operating system product is CP/NOS[T.M.]. This product is for applications where the requester microcomputer does not have any disk resources and is therefore unable to run CP/M. CP/NOS consists of a bootstrap loader that can be placed into ROM or PROM, a skeletal CP/M that contains only the console and printer functions, and the logical and physical portions of the CP/NET requester. At the user level, CP/NOS provides a virtual CP/M 2.X system to the requester microcomputer. A requester microcomputer can consist of simply a processor, memory, and an interface to the network. Thus, a CRT with sufficient RAM can execute CP/M programs, performing its computing locally while depending on the network to provide all disk, printer, and other I/O facilities. The following figure illustrates the relationship between CP/NOS, MP/M, and CP/NET.

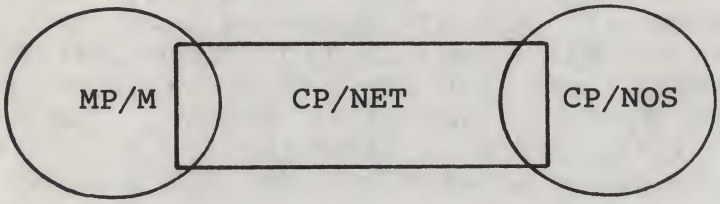


Figure 1-2. CP/NOS Configuration

MP/NET[T.M.], a third network operating system product, allows MP/M systems to share each others resources on the network. With MP/NET, there is no distinction between a server and a requester because all the nodes on an MP/NET can manage shared resources and initiate network messages. Thus, MP/NET provides a symetrical network where all the nodes have equal capability. The following figure illustrates the relationship between MP/M and MP/NET.

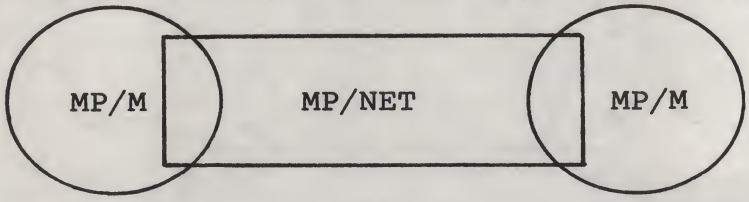


Figure 1-3. MP/NET Configuration

The three products described above, CP/NET, CP/NOS, and MP/NET, can be combined to produce a composite network consisting of MP/M servers that can share each others files, CP/M requesters, and diskless CP/NOS requesters.

1.1 Functional Description of CP/NET

CP/NET operates in multiple processor environments that range from tightly to loosely coupled processors. In this document, tightly coupled processors are defined as processors sharing all or a portion of common memory. Communication of interprocessor messages is at memory speed. Loosely coupled processors are those that do not have access to memory that is common or accessible by both processors. Communication between loosely coupled processors can be implemented with a serial data link or possibly a high-speed parallel bus.

The CP/NET operating system is an upward-compatible version of CP/M 2.X, that provides selected system I/O facilities to requester microcomputers via a network. Network access to system I/O facilities is provided by additions to the Basic I/O System (BIOS) called the Slave Network I/O Systems (SNIOS) and a new Basic Disk Operating System (BDOS) called the Network Disk Operating System (NDOS). The requester NDOS and NIOS are loaded and executed while running under CP/M 2.X.

In addition to the standard CP/M facilites, CP/NET provides the following capabilities:

- The network can be accessed for system I/O facilities.
- Messages can be transmitted and received between requesters and servers.
- With an electronic mail system, requesters and servers can send mail to each other.

The MP/M server is implemented by adding some resident system processes at system generation (GENSYS) time. The resident system processes include the slave support processes (SLVSP), provided by Digital Research to perform the logical message handling functions for the server, and the network interface processes (NETWRKIF), customized by you for a particular hardware network interface.

1.1.1 CP/NET Configurations

The following figures illustrate possible CP/NET configurations. The interprocessor message format permits multiple CP/NET servers, so that if the hardware capability exists, more than one server can be present in a network.

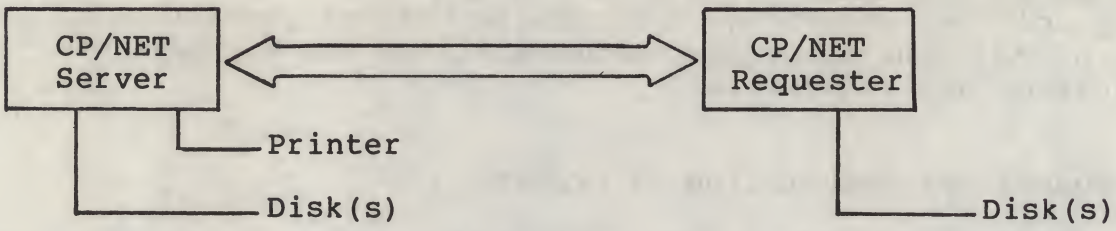


Figure 1-4. Server and Single Requester

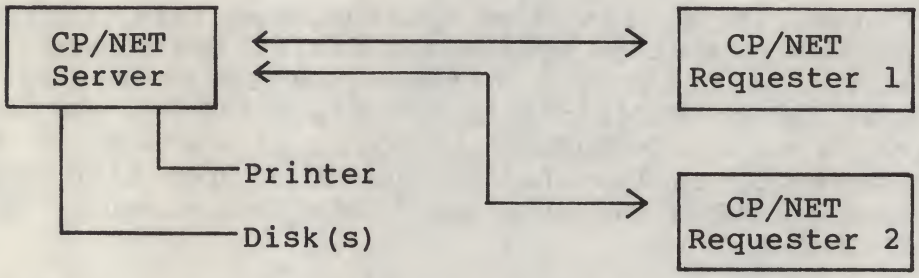


Figure 1-5. Active Hub Star Configuration

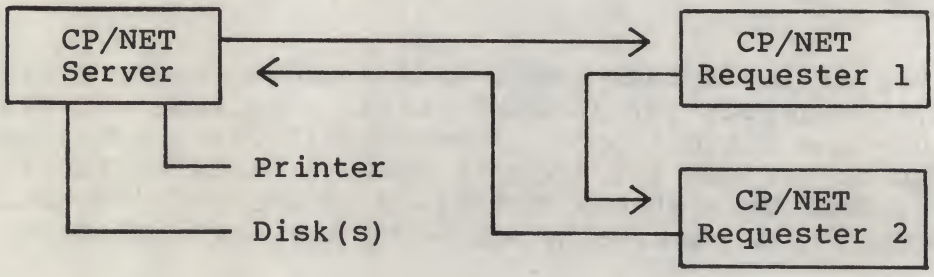


Figure 1-6. Ring Configuration

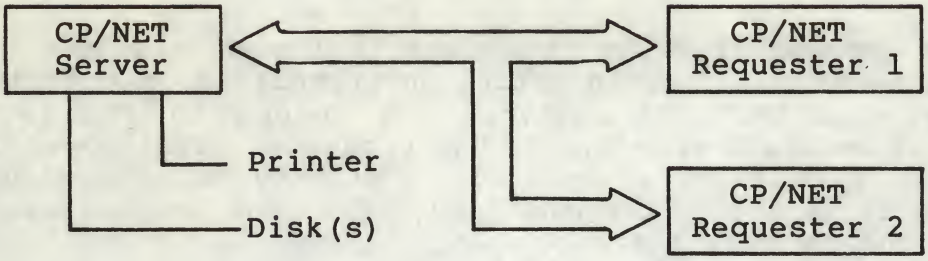


Figure 1-7. Bus Configuration

1.1.2 Hardware Environment

The hardware environment for CP/NET must include two or more microcomputers that have some means of interprocessor communication.

One of the microcomputers must execute the MP/M operating system to provide the CP/NET server facilities. The processor executing MP/M must be an 8080, 8085, or Z80 CPU with a minimum of 32K bytes of memory, 1 to 16 consoles, 1 to 16 logical or physical disk drives each containing up to eight megabytes, a clock/timer interrupt, and a network interface.

The CP/NET requester microcomputers must have 8080, 8085, or Z80 CPUs with at least 16K bytes of memory , 0 to 16 logical or physical disk drives each containing up to eight megabytes, and a network interface. A console is not absolutely required although it is strongly recommended.

While not an absolute requirement, it is strongly recommended that requester processors have and control their own console devices because typical microcomputer applications require significant amounts of console I/O. Thus, without a local console of its own, a requester places a very large demand upon the network and the CP/NET server.

1.2 Requester Commands

This section describes the requester commands that enable you to access the network and use the resources on the network. All the requester commands are actually COM files that reside on disk at the requester.

1.2.1 LOGIN

The LOGIN command allows a requester to log in to a specified server. A requester must log in before any resources on the server can be accessed. Once a requester has logged in, it is not necessary to log in again even though the requester might power down and then power up again. A requester can only be logged off a server by an explicit LOGOFF command issued from the requester. The command takes the general form:

LOGIN {password}{[mstrID]}

where: password is an optional 8 ASCII character password; the default password is PASSWORD.

[mstrID] is an optional 2 digit server processor ID; the default is [00].

The following is the most simple form:

A>**LOGIN**

1.2.2 LOGOFF

The LOGOFF command allows a requester to log off from a specified server. Once a requester has logged off, the server cannot be accessed again until you issue a LOGIN command. The command takes the general form:

LOGOFF {[mstrID]}

where: [mstrID] is an optional 2 digit server processor ID; the default is [00].

The following is the most simple form:

A>**LOGOFF**

1.2.3 SNDMAIL

The SNDMAIL command allows a requester to send mail to another requester or server. The command takes the general form:

SNDMAIL {[mstrID]} (destID) "message to be sent"

where: [mstrID] is an optional 2 digit server processor ID; the default is [00].

(destID) is the 2 digit destination processor ID.

"message" is any message enclosed in quotation marks.

The following is the most simple form:

A>**SNDMAIL (12) "Hello"**

1.2.4 RCVMAIL

The RCVMAIL command allows a requester to obtain all the mail posted for him by a specified server. The command takes the general form:

RCVMAIL {[mstrID]}

where: [mstrID] is an optional 2 digit server processor ID; the default is [00].

The following is the most simple form:

A>**RCVMAIL**

1.2.5 NETWORK

The NETWORK command enables a requester to assign selected I/O to the network. The NETWORK command updates the requester configuration table. The command takes the general form:

NETWORK {local dev} {=} {server dev}{[mstrID]}

where: local dev is the specification of a local device such as LST: A:,... CON:

server dev is the specification of a server device such as A:,...

[mstrID] is an optional 2 digit server processor ID: the default is [00];

The following are some typical assignments:

A>**NETWORK LST:**

A>**NETWORK LST:=3[07]** (list dev #3 on server 07)

A>**NETWORK CON:=2** (console #2 on dflt mstr)

A>**NETWORK B:=D:** (logical B: is servers D:)

1.2.6 LOCAL

The LOCAL command enables a requester to reassign selected I/O back to local from the network. The LOCAL command updates the requester configuration table. The command takes the general form:

LOCAL {local dev}

where: local dev is the specification of a local device such as: LST:, A:,... CON:

The following are some typical assignments:

A>**LOCAL LST:**

A>**LOCAL B:**

1.2.7 ENDLIST

The ENDLIST command simply sends the control-Z (1Ah) character to the list device. This command terminates list outputs to the network.

A>**ENDLIST**

1.2.8 DSKRESET

The DSKRESET command is identical in function to the PRL that executes under MP/M. It resets the specified drive, so that a disk can be changed. The command takes the general form:

DSKRESET {drive(s)}

where: drive is a list of the drive names to be reset.

The following are some typical disk resets:

A>**DSKRESET** (resets all drives)

A>**DSKRESET B:,F:** (reset drive B: and F:)

1.2.9 CPNETLDR

The CPNETLDR command loads the requester CP/NET system. Specifically, the SNIOS.SPR file loads and relocates directly below the CP/M BDOS, and the NDOS.SPR file loads and relocates directly below the SNIOS.

From that point on, the BIOS, BDOS, SNIOS, and NDOS remain resident in memory. The CPNETLDR does not require any user customization. It displays an error message if loader errors are encountered. The following example shows a typical CPNETLDR execution.

```
A>CPNETLDR

CP/NET 1.1 Loader
=================
BIOS          F600H  0A00H
BDOS          E800H  0E00H
SNIOS   SPR   E500H  0300H
NDOS    SPR   DB00H  0A00H
TPA           0000H  DB00H

CP/NET 1.1 loading complete.
NDOS initialization complete.
<Warm Boot>
A>
```

1.2.10 CPNETSTS

The CPNETSTS command displays the requester configuration table. The requester configuration table indicates the status of each logical device, that is either local or assigned to a specific server on the network. The following example shows a typical CPNETSTS execution.

```
A>cpnetsts

CP/NET 1.1 Status
================
Requester processor ID = 34H
Network Status Byte = 10H
Disk device status:
  Drive A: = LOCAL
  Drive B: = LOCAL
  Drive C: = Drive A: on Network Slave ID = 00H
  Drive D: = Drive B: on Network Slave ID = 00H
  Drive E: = LOCAL
  Drive F: = LOCAL
  Drive G: = LOCAL
  Drive H: = LOCAL
  Drive I: = LOCAL
  Drive J: = LOCAL
  Drive K: = LOCAL
  Drive L: = LOCAL
  Drive M: = LOCAL
  Drive N: = LOCAL
  Drive O: = LOCAL
  Drive P: = LOCAL
Console Device = LOCAL
List Device = List #0 on Network Slave ID = 00H
A>
```

1.2.11 CONTROL-P

A CTRL-P causes console output to be echoed to the list device until the next CTRL-P. The messages CTL-P ON and CTL-P OFF are displayed at the console. When the requester list device has been networked, the local system is now using the server printer, for example, the second CTRL-P causes a CTRL-Z (ASCII 1AH) to be sent to the server, causing the server to close and print the spool file.

Note: when the requester is using the server printer by invoking with a CTRL-P, the requester must issue a second CTRL-P to cause the server to close the spooled file and begin printing it. When the requester is using the server printer and has invoked it with a program such as PIP, the warm boot that occurs at program termination causes the required CTRL-Z to be sent to the server to close and print the spooled file.

The program ENDLIST is no longer needed to terminate network list output in these situations.

1.3 Server Commands

This section describes the server commands that enable the operator of an MP/M server to access network mail. This section also describes a spooler that spools and deletes files. The requester commands to access the network mail are actually PRL files that reside on disk at the server.

1.3.1 BROADCST

The BROADCST command allows a server to broadcast a piece of mail to all the logged in requesters. The command takes the general form:

```
BROADCST "message to be sent"
```

where: "message" is any message enclosed in quotation marks.

The following is a typical broadcast:

```
0A>BROADCST "Printer #1 will be taken off-line at 14:00"
```

1.3.2 MRCVMAIL

The MRCVMAIL command allows a server to obtain all the mail posted for him. The command takes the general form:

```
MRCVMAIL
```

1.3.3 MSNDMAIL

The MSNDMAIL command allows a server to send mail to another requester or server. The command takes the general form:

MSNDMAIL {[mstrID]} (destID) "message"

where: [mstrID] is an optional server processor ID; the default is [00].

(destID) is the destination processor ID.

"message" is any message enclosed in quotation marks.

The following is the most simple form:

0A>**MSNDMAIL (12) "Hello"**

End of Section 1

Section 2
CP/NET Requester Interface Guide

This section describes the logical portion of the CP/NET requester and the CP/NET system organization including the memory structure, the message format for interprocessor communication, the recommended RS-232C point-to-point protocol, and the Network Disk Operating System (NDOS) function calls.

The requester portion of CP/NET is logically divided into four modules: the Basic I/O System (BIOS), the Basic Disk Operating System (BDOS), the Slave Network I/O System (SNIOS), and the Network Disk Operating System (NDOS). The BIOS and BDOS are unmodified from CP/M and are described in the appropriate CP/M documentation. The SNIOS is a hardware-dependent module that defines the low level interface to the NDOS that is necessary for network I/O. Although Digital Research supplies a standard SNIOS, Section 3 provides explicit instructions for field reconfiguration of the SNIOS to match nearly any hardware network environment.

The NDOS intercepts all CP/M BDOS function calls and determines if the operation is to be performed locally or on the network. If the operation is local, control is transferred to the BDOS. If the operation is to be done on the network, the NDOS forms the appropriate logical message, and then sends it to the server to perform the specified function via the SNIOS. After sending the function to the server, the requester waits for a response. Thus, there is a full handshake between the requester and the server for each function that the requester makes of the server.

The following figure shows CP/NET requester memory structure:

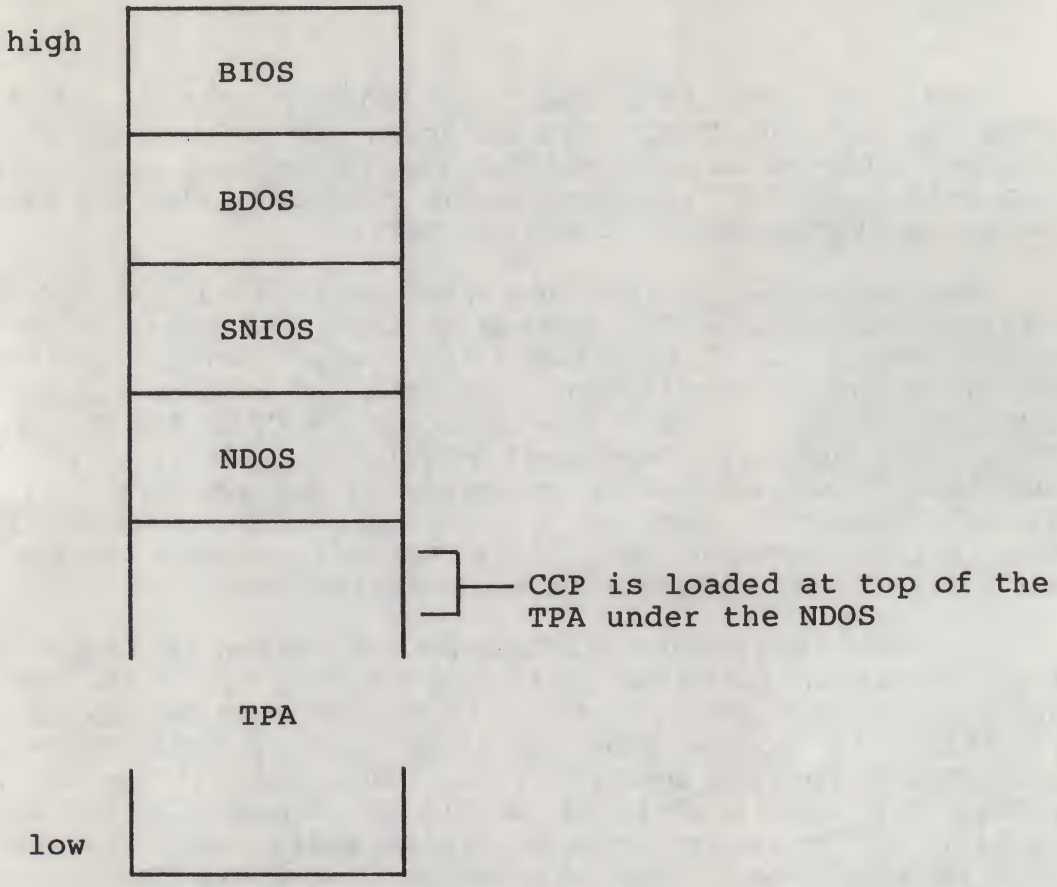


Figure 2-1. CP/NET Requester Memory Structure

The CPNETLDR program loads and relocates the SNIOS and NDOS underneath the BDOS. From that point on, the SNIOS and NDOS remain in memory, trapping warm and cold boots that simply load and relocate the CCP at the top of the TPA under the NDOS.

2.1 CP/NET Interprocessor Message Format

The simple message format that CP/NET uses for interprocessor communication includes some packaging overhead and the actual message itself. The packaging overhead consists of a message format code, a CP/NET destination address, a CP/NET source address, a CP/M function code, and a message size, defined as follows.

Message Format Code

The message format code is a single byte that specifies the format of the message itself. Message formats 0 - 127 are reserved by Digital Research for general interprocessor message format codes. The general interprocessor format codes follow the message format shown below, but differ in length of the individual fields (Appendix A).

The odd-numbered format codes, least significant bit is a 1, are for response messages sent back from servers to requesters. Thus, a CP/M disk read function sent from a requester to a server has a message format code of 0, and the return code sent back from the server to the requester has a message format code of 1.

You should implement the general interprocessor message formats 0 and 1 as shown in Appendix A because they enable interconnection among microcomputers from different vendors.

Message Destination Processor ID

The message destination processor ID field is generally one byte long. However, some message formats specify a two-byte field.

A destination field value of 0 is a special case. It specifies the local network level server as the destination processor. This destination address is particularily useful in systems where a multilevel network has been implemented, in which each requester can only directly access one server, and the requesters do not necessarily know the address of the server at their network level.

Message Source Processor ID

The message source processor ID field is generally one byte long. However, some message formats specify a two-byte field.

CP/M Function Code

The CP/M function code field is one byte long. The size of the message data field generally depends upon the CP/M function. Each CP/M function has a specific number of bytes to be sent to the server and a specific number of bytes to be returned to the requester. Appendix E provides the logical message specification for each of the CP/M functions. Some of the CP/M function codes have no network equivalent function.

Size

The size field is generally one byte long. However, some message formats specify a two-byte field. The size value has a bias of 1. Thus, a size of 0 specifies an actual size of 1, while a size of 255 specifies an actual size of 256. With a one byte size field, the minimum data field is 1 byte while the maximum is 256.

The message format shown in the following figure, does not contain a cyclic redundancy code (CRC) or any other error checking as a part of the packaging overhead. The user written SNIOS can add the error checking when it actually places the message onto the network and then test the message when the SNIOS receives a message from the network. This function is intentionally left to the user, avoiding redundant error checking where standard interface protocols, both in software and hardware, might already provide error checking.

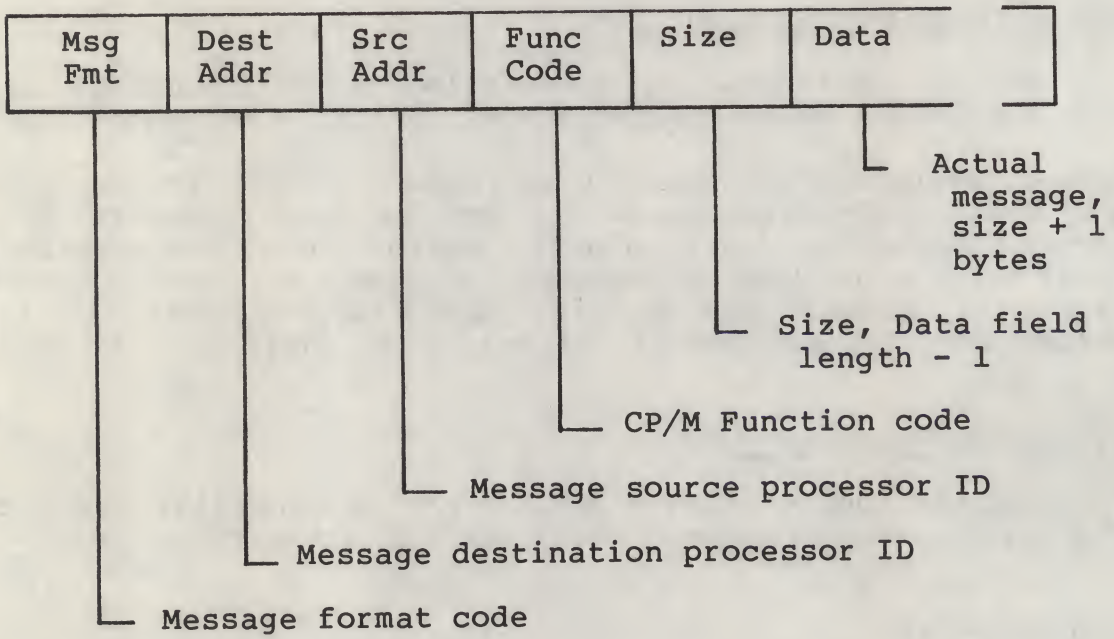


Figure 2-2. Message Format

2.2 Recommended RS-232C Point-to-Point Protocol

Digital Research developed a relatively simple RS-232C point-to-point protocol to provide a demonstration vehicle for CP/NET and to encourage compatibility among the hardware vendors. The protocol, as implemented in the sample SNIOS and NETWRKIF, breaks the logical message into a fixed header and a variable length data portion whose size is obtained from the fixed header. This

simplifies operation with DMA channels that need terminal counts and also provides a checksum for the header that contains the SIZ field.

As local network standards emerge, possibly from the IEEE, we will provide other protocols as appendixes to this manual or as CP/NET application notes. It is important to realize that CP/NET does not require any specific protocol because of its logical message structure. However, a protocol is recommended when connecting microcomputers from different vendors.

Appendix B contains the recommended server-requester handshake for RS-232C, and appendixes C and D illustrate the application of the protocol to 8-bit binary and 7-bit ASCII configurations.

2.3 Network Disk Operating System Functions

The Network Disk Operating System (NDOS) is an extension of the CP/M 2.X Basic Disk Operating System (BDOS). The CPNETLDR program loads and excutes the NDOS portion of CP/NET on a running CP/M system. NDOS allows selected system I/O functions to be performed on a CP/NET server via the network rather than on the requester itself. For example, list output can be directed to a specific server on the network. The NDOS also permits a mix of local and network logical or physical disk drives. In such applications, you can specify the mix of drives at initialization using the configuration table, or at run-time using the NETWORK and LOCAL commands.

The NDOS intercepts all the CP/M calls made to location 0005H in the requester and then uses the requester configuration table located in the SNIOS to determine whether the function is to be performed locally or on the network.

Several network disk operating system functions have been added to the CP/M BDOS functions. The following paragraphs describe the entry parameters and returned values for each NDOS function.

To gain access to the NDOS functions, pass a function number and information address through the primary entry point at location 0005H. In general, the function number is passed in register C with the information address in the double register pair DE. Single-byte values are returned in register A, with double-byte values returned in HL. A 0FFH value is returned when the function number is out of range. The register passing conventions of NDOS agree with those of Intel® PL/M™ systems programming language.

Note: a returned value of FFH can also indicate a network interface failure. Such failures can be detected by obtaining the network status byte (Function 68).

FUNCTION 64: LOGIN
Entry Parameters: Register C: 40H DE: Login Msg Adr Returned Value: Register A: Return Code

The LOGIN function allows a requester to communicate with a specified server. The passed parameter, in the DE register pair, is the address of the login message. The login message consists of a one-byte server ID, followed by an 8-character ASCII field containing a password.

The return code sent back in register A indicates whether or not the login was successful. A value of 0 indicates a successful login while an FFH indicates login failure. Failure could result because the maximum number of requesters for the server to support have already logged in, or because of an incorrect password.

A passed server processor ID of zero specifies the direct server of the requester. It is used when the server processor ID is unknown and when the requester has only a single server. For example:

```
LOGIN   EQU     64
        ...
        MVI     C,LOGIN
        LXI     D,Loginmsg
        CALL    BDOS            ;login to network
        ...

Loginmsg:
        DB      00h             ;default server ID
        DB      'Secret  '      ;password
```

FUNCTION 65: LOGOFF
Entry Parameters: Register C: 41H E: Master Proc. ID Returned Value: Register A: Return Code

The LOGOFF function indicates that a requester no longer requires the services of the server. A logoff frees the requester support resident system process and makes this available to another requester wanting to login. The passed parameter, in register E, is the server processor ID. A return code of 0 indicates a successful logoff, while a FFH indicates failure. A failure usually occurs if the requester was not logged into the server in the first place. For example:

```
LOGOFF  EQU     65
        ...
        MVI     C,LOGOFF
        MVI     E,07h           ;server ID = 07h
        CALL    BDOS            ;logoff from network server
        ...
```

FUNCTION 66: SEND MESSAGE ON NETWORK
Entry Parameters: Register C: 42H Registers DE: Message Address Returned Value: Obtained from a subsequent call to receive msg, first byte of message data field.

The SEND MESSAGE ON NETWORK function sends the message stored in memory at the location given by DE to the destination processor on the network. This function can send any function to the network because it provides direct access to the SENDMSG entry point in the SNIOS.

This function can send electronic mail to another requester or server on the network by specifying SEND MESSAGE ON NETWORK as the FNC field of the message.

Note: immediately following every message sent on the network, you must receive the acknowlege message back from the server using the receive message function.

For example, after sending a SEND MESSAGE ON NETWORK logical message to a server, you must make a RECEIVE MESSAGE FROM NETWORK call to get the return code back from the call. The returned value is a return code that is 0 if the SEND MESSAGE ON NETWORK logical message was successfully posted at the server, or an FFh if unsuccessful. Failure to post the message results either from the requester processor not being logged in, or a full mailbox at the server. For example:

```
SNDMSG  EQU     66
RCVMSG  EQU     67
        ...

        MVI     C,SNDMSG
        LXI     D,Letter
        CALL    BDOS            ;send message on network
        MVI     C,RCVMSG
        LXI     D,Letter
        CALL    BDOS            ;receive acknowledge
                                ; from server, msg.msg(0)
                                ; will get return code
        ...

Letter:
        DB      00h             ;msg.fmt
        DB      00h             ;msg.did
        DB      01h             ;msg.sid
        DB      SNDMSG          ;msg.fnc
        DB      10               ;msg.siz
        DB      02h             ;msg.msg(0)   (message dest)
        DB      'Hello Gary'    ;msg.msg(1) - msg.msg(10)
```

FUNCTION 67: RECEIVE MESSAGE FROM NETWORK
Entry Parameters: Register C: 43H Registers DE: Message Buffer Address

The RECEIVE MESSAGE FROM NETWORK function receives a message from the network and places it into the message buffer addressed by registers DE. This function provides direct access to the RECEIVEMSG entry point in the SNIOS.

Use this function to receive electronic mail from another requester or server on the network by specifying RECEIVE MESSAGE FROM NETWORK as the FNC field of a message sent on the network, and then making the BDOS RECEIVE MESSAGE FROM NETWORK call. For example:

```
SNDMSG  EQU     66
RCVMSG  EQU     67

        ...

        MVI     C,SNDMSG
        LXI     D,Letter
        CALL    BDOS            ;send message on network
                                ; contains receive msg fnc
        MVI     C,RCVMSG
        LXI     D,Letter
        CALL    BDOS            ;receive message from ntwrk

        ...


Letter:
        DB      00h             ;msg.fmt
        DB      00h             ;msg.did
        DB      01h             ;msg.sid
        DB      RCVMSG          ;msg.fnc
        DB      0               ;msg.siz
        DS      1               ;msg.msg(0)  (for src ID)
        DS      255             ;msg.msg(1)-msg.msg(255)
```

FUNCTION 68: GET NETWORK STATUS	
Entry Parameters:	
Register	C: 44H
	E: Processor ID
Returned Value:	
Register	A: Network Status

The GET NETWORK STATUS function returns the network status byte defined in Section 3.1.

FUNCTION 69: GET CONFIGURATION TABLE ADDRESS	
Entry Parameters:	
Register	C: 45H
Returned Value:	
Registers	HL: Table Address

The GET CONFIGURATION TABLE ADDRESS function returns the address of the configuration table maintained in the user-written SNIOS. Various system programs use and alter the information provided in the configuration table to redirect I/O to the network. The configuration table for CP/NET requesters is described in Section 3.2.

End of Section 2

Section 3
CP/NET Requester Alteration Guide

CP/NET can support 1 to 16 requesters per server. Each CP/NET requester consists of code that is dependent on the physical environment, network interface, and an operating system (NDOS) that is capable of using the network to perform system I/O.

The code that performs the physical network interface functions for the CP/NET requester resides in a user-supplied module named the Slave Network I/O System (SNIOS). You prepare the SNIOS in system page relocatable format and in a file of type SPR. The SNIOS is loaded into memory by the CPNETLDR program and is then used by the NDOS.

This section assists you in customizing the SNIOS. Included is a discussion of the SNIOS entry points, the requester configuration table, and the steps to implement and debug a custom SNIOS. Appendix G provides a listing of a sample SNIOS.

3.1 Slave Network I/O System Entry Points

The SNIOS begins with a jump vector containing the network I/O system entry points as shown below:

```
NIOS+00H   JMP  NETWORKINIT   ; Network initialize
NIOS+03H   JMP  NETWORKSTS    ; Rtn network status
NIOS+06H   JMP  CONFIGTBLADR  ; Rtn Config. Tbl Adr
NIOS+09H   JMP  SENDMSG       ; Send msg on network
NIOS+0CH   JMP  RECEIVEMSG    ; Receive msg from ntwk
NIOS+0FH   JMP  NTWRKERROR    ; Network error
NIOS+12H   JMP  NTWRKWBOOT    ; Network warm boot
```

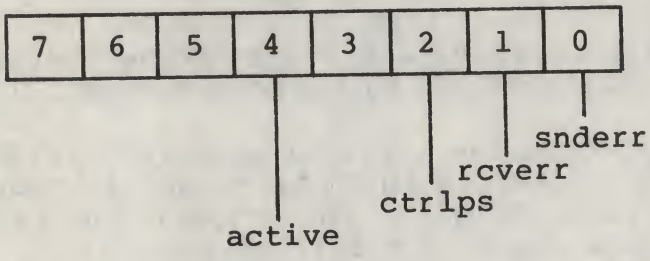
Each jump address corresponds to a particular subroutine that performs the specific function. The exact responsibilities of each entry point subroutine are given below.

NETWORKINIT

This SNIOS entry point is called when control is transferred to the NDOS initialization entry point after being loaded by the CPNETLDR. This subroutine performs any required network interface intialization. Initialization may include reading back-panel switches, or some other suitable source, to obtain the requester processor ID for the configuration table.

NETWORKSTS

This subroutine returns a single byte in register A and determines the status of the network interface. The error bits are to be reset when the call is made. The format of the network status byte is shown below:



where, active = 1 if requester logged in

ctrlps = 1 if control P is active

rcverr = 1 if error in received message

snderr= 1 if error in sending a message

CONFIGTBLADR

This subroutine returns the configuration table address in the HL register pair.

SENDMSG

This subroutine enables messages to be sent from one processor to another via the network. The passed parameter, in registers BC, is a pointer to the message. Control is not returned from this procedure until the message has been sent. Thus, the message pointed to by the BC register pair can be modified immediately upon return. The return code, in register A, has either a value of 0 indicating success, or FFH indicating failure to access the network.

RECEIVEMSG

Messages are received from another processor through the network with this subroutine. The passed parameter, in registers BC, is a pointer to a message buffer. Control is not returned from this procedure until the message has been received and placed into the message buffer. Thus, the message in the buffer is valid immediately upon return. The return code, in register A, has either a value of 0 indicating success, or FFH indicating failure to access the network.

NTWRKERROR

When network errors are encountered, this procedure is called. Any required network interface device reinitialization should be performed. In typical SNIOS implementations, executing a return from the NTWRKERROR procedure results in a retry. If a retry is not wanted, an appropriate message is displayed on the console and a warm boot is performed.

NTWRKWBOOT

This SNIOS procedure is called each time the NDOS reloads the CCP. The sample SNIOS displays a <Warm Boot> message on the console only as a demonstration of NTWRKWBOOT. More practical applications of this procedure include interrogating the CP/NET server for messages. In this way, each time a warm boot is performed, any messages posted for the requester can be displayed on the console.

3.2 Requester Configuration Table

The configuration table that resides in the CP/NET requester SNIOS allows reassignment of physical and logical devices. The configuration table creates a mapping of logical to physical devices that can be altered during CP/NET processing. In particular, the configuration table specifies the system I/O to be accessed through the network.

The requester configuration table is defined as follows:

000-000 Requester status byte

001-001 CP/NET requester processor ID

002-033 Disk Devices, 16 two-byte pairs, first byte high-order bit on = drive on network with the server drive code in the least significant 4 bits, the second byte contains the server processor ID

034-035 Console Device, first byte high-order bit on = console I/O on network with the server console number in the least significant 4 bits, the second byte contains the server processor ID

036-037 List Device, first byte high-order bit on = list to network with the server list device number in the least significant 4 bits, the second byte contains the server processor ID

038-038 List Device buffer index

039-043 List Device logical message header: FMT, DID, SID, FNC and SIZ

044-044 List Device server list device number

045-172 List Device buffer

3.3 Implementing and Debugging a Custom SNIOS

The following steps indicate how to implement and debug a custom SNIOS.

1) Obtain assembled listings of the SNIOS.ASM source file that require modification. You can use MAC[T.M.], RMAC[T.M.], or ASM[T.M.]. If you use ASM, the title, name, if and else statements must be removed from the source files to assemble correctly. It is highly recommended that you use RMAC because it simplifies the task of generating the SPR files when used in conjunction with LINK[T.M.]. Otherwise, the SPR files have to be generated in the same manner as for MP/M XIOS.SPR generation.

   A>**RMAC SNIOS**

2) Study the SNIOS.PRN listing. ASCII equ; if true it specifies that the message format is 7-bit ASCII, if false it specifies a binary 8-bit message format. The ASCII mode is sometimes useful in debugging, but in practice it should not be used where it is possible to transmit 8-bit serial data.

   The only code that requires modification in the SNIOS.ASM file is contained in the Charout, Charin, and Delay procedures. The Charout and Charin procedures can be conditionally assembled for either a Dynabyte DB8/2™, a Digital Microsystems DSC-2™, or an ALTOS 8000-2. The NOPs in the Charout procedure are simply padding, so that the length of the DB8/2 SNIOS and DSC-2 SNIOS is the same. It saved making an extra listing.

   Note: the Charin procedure contains a timeout loop that might have to be modified according to the requester processor speed. This is also true of the Delay procedure.

3) Prepare the SNIOS.SPR file as shown below:

```
A>RMAC SNIOS
A>LINK SNIOS[OS]
```

   The output of the linker is the SNIOS.SPR file.

   If you do not use RMAC and LINK, use ASM, PIP, and GENMOD as shown below:

```
A>ASM SNIOS
    ;assemble with ORG 0000H

A>REN SNIOS0.HEX=SNIOS.HEX
    ;edit SNIOS.ASM ORG statement

A>ASM SNIOS
    ;assemble with ORG 0100H

A>REN SNIOS1.HEX=SNIOS.HEX

A>PIP SNIOS.HEX=SNIOS0.HEX,SNIOS1.HEX
    ;concatenate the HEX files

A>GENMOD SNIOS.HEX SNIOS.SPR
    ;generate the SNIOS.SPR file
```

4) Copy the following files to the requester:

```
CPNETLDR.COM = Loads CP/Net (NDOS.SPR and SNIOS.SPR)
CPNETSTS.COM = Displays status of the system I/O
NETWORK.COM  = Redirects I/O from local to network
LOCAL.COM    = Redirects I/O from network to local
DSKRESET.COM = Resets specified logical drives
```

```
LOGIN.COM    = Logs on to server
LOGOFF.COM   = Logs off from server
SNDMSG.COM   = Send message to specified requester or server
RCVMSG.COM   = Receive message
NDOS.SPR     = Network Disk Operating System
SNIOS.SPR    = Custom Slave Network I/O System
CCP.SPR      = Console Command Processor
```

5) The SNIOS can be debugged in a manner similar to MP/M, as follows:

```
A>DDT CPNETLDR.COM

*IB
*sl03
0l03 07 xx ;where xx is the restart the debugger uses
           ; usually 7
*g
```

At this point, CP/NET loads, displaying the memory map, and then breaks at the specified restart.

You can place breakpoints at desired locations, and then issue a G command that specifies the address following the restart instruction where the CPNETLDR broke.

Perhaps the most critical area that requires adjustment of the SNIOS for a specific network configuration is in the timeout code of the Charin procedure. If too short a time is allowed, the server might not be able to complete the function because of a heavy request load from the requesters. If too long a time is specified, communication breaks on the network can go undetected for a period of time and make both error recovery and precise detection difficult.

Another critical parameter that requires adjustment for different environments is ALWAYS$RETRY. This Boolean, when true, controls conditional assembly that always produces retries on network failures. In this mode of operation, it is possible to recover from broken communication between the requester and a server. However, it does hang the requester in a busy retry mode when failures occur.

End of Section 3

Section 4
CP/NET Server Alteration

CP/NET must have one or more servers. Each CP/NET server consists of resident system processes running under the MP/M operating system and code that is dependent on the physical environment, network interface.

This section contains information that allows you to write a custom Network Interface Process (NETWRKIF). It includes a discussion of the server network I/O system, the server configuration table, and the steps required to implement and debug a custom NETWRKIF.

4.1 Server Network I/O System

The code that performs the physical network interface functions for the CP/NET server resides in a user-supplied module named the Network Interface Process (NETWRKIF). You prepare the NETWRKIF in the form of a Resident System Process (RSP). It is included at GENSYS time under MP/M and then loaded as part of the MPM.SYS file into memory by the MPMLDR.

4.2 Network Interface Process

The network interface processes are part of the user-written NETWRKIF module. They perform the actual physical I/O for the CP/NET server. There is typically one network interface process per requester that the server supports. The names of the requester network interface process descriptors range from NtwrkIP0 to NtwrkIPF where the last character in the process descriptor name is a single Hex-ASCII digit in the decimal range of 0 to 15.

Queues pass messages between the interface processes and the requester support processes. The requester support processes are provided for the CP/NET server in the form of a resident system process (Section 4.3).

Two queues per requester supported by the server are created during NETWRKIF initialization: one for message input and one for message output. The queue names for input, network queue in, range from NtwrkQI0 to NtwrkQIF. The queue names for output, network queue out, range from NtwrkQO0 to NtwrkQOF.

The following figure illustrates the interaction between the slave support processes and the network interface processes that actually handle the direct physical I/O between the server and the requesters.

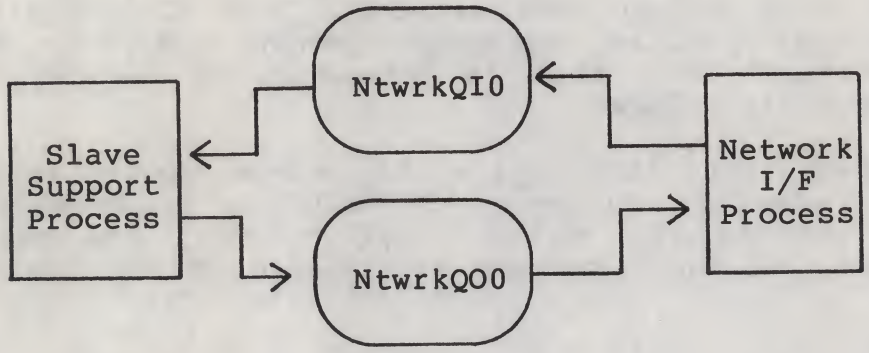


Figure 4-1. Relationships Between Processes

The messages passed between the network interface processes and the slave support processes are two-byte pointers to the actual message received from the network or to be sent to the network.

4.3 Slave Support Processes

The CP/NET server supports one process per requester. The slave support processes all execute the same reentrant piece of code. They perform the function specified by the message received from the requester through the network. The slave support process names range from Slave0SP to SlaveFSP.

An FCB table is maintained for each requester. Specific FCBs are identified by FCB addresses in the server. For example, a CP/M open file function message received from a requester through the network includes drive, filename, filetype, and extent. The server returns the address of the entry in the server FCB table. Only the exact amount of information required to perform the function is transmitted on the network. The actual FCB for file I/O is prepared in the FCB table in the server and used when the open is performed by the server.

4.4 Server Configuration Table

The configuration table that resides in the CP/NET server NETWRKIF provides information about the network. The server uses information during initialization to set up the requester support processes.

The configuration table for servers is defined as follows:

000-000	Server status byte
001-001	Server processor ID
002-002	Max number of requesters to be supported (1-16)
003-003	Number of logged in requesters
004-005	16-bit vector of logged in requesters
006-021	Requester processor IDs, one byte per processor
022-029	Login password
030-	Requester support process descriptors and stacks

4.5 Implementing and Debugging a Custom NETWRKIF

The steps to implement and debug a custom NETWRKIF are as follows:

1) Obtain an assembled listing of the NETWRKIF.ASM source file that requires modification. Use MAC, RMAC, or ASM. If you use ASM the title, name, and if and else statements must be removed from the source files to assemble correctly. It is highly recommended that you use RMAC because it simplifies the task of generating the *.SPR files when used in conjunction with LINK. Otherwise, the *.SPR files have to be generated in the same manner as for MP/M XIOS.SPR generation.

   A>**RMAC NETWRKIF**

2) Study the NETWRKIF.PRN listing. This initial version of the server network I/F module is by no means optimal, but it is portable. That is, it should be a very simple matter to customize the NETWRKIF.ASM file for any CP/NET server running MP/M. There are several areas of the NETWRKIF.ASM that require modification. They are the network interface process descriptors and their associated queues and buffers; the server configuration table, containing storage for the requester support process descriptors and stack; the mail box data structures that specify the number of pieces of mail, the mail size, and buffer area; the watchdog timer process, in particular the table specifying the flag that is to be set at a timeout; and the local data segment that contains several tables used to drive the requester interfaces.

3) Modify the NETWRKIF.ASM file and produce the NETWRKIF.RSP file:

- The Network interface process descriptors and their associated queues and buffers:

  The sample NETWRKIF.ASM contains support for 4 requesters. This number can be modified by making the changes in this section of the file as follows. Duplicate the following data structures for each additional requester:

```
NtwrkIP1       = Process descriptor, link points to the
                 next requester process descriptor.
NtwrkIS1       = Stack area, return address is init.
QCBNtwrkQI1    = QCB for network queue input.
UQCBNtwrkQI1   = User QCB for network queue input.
BufferQI1Addr  = Buffer address for queue input.
QCBNtwrkQO1    = QCB for network queue output.
UQCBNtwrkQO1   = User QCB for network queue output.
BufferQO1Addr  = Buffer address for queue output.
BufferQ1       = Buffer.
```

- The server configuration table must indicate the number of requesters supported, and it must contain space for each requester support process descriptor and stack.

  The default password is PASSWORD and is contained in the configuration table. This default is also generated by the LOGIN program.

- The Mail Box data structures are set up for a maximum of 4 requesters, with 4 pieces of mail each 128 bytes in length. Any of these parameters can be easily modified.

- The Watchdog timer process might require both code and table changes. The watchdog table specifies a flag to be set in the event of a timeout while waiting for character input from the network. This flag number must correspond to the flag on which the XIOS character input routine is waiting. Polled I/O operation for the server network interface is not recommended.

  It is assumed in the NETWRKIF that a 16ms tick has been implemented in the MP/M XIOS. Ticks at different rates affect the timeout period. The omission of ticks from an MP/M XIOS does not allow CP/NET to execute.

- The Local data segment contains tables for BinaryASCII, delaycounts, and chariotbl. Each of these tables must be expanded to accommodate the specific number of requesters.

  The most critical table is chariotbl. This table establishes the relationship between the requesters and the console device number in the MP/M console XIOS. The NETWRKIF module uses direct XIOS console calls for I/O to the physical requester interface. This is the way simple portability was achieved. If the 8-bit mode is used, the XIOS character handler must not mask the high-order bit, normally parity.

4) Prepare the NETWRKIF.RSP file as shown below:

```
A>RMAC NETWRKIF

A>LINK NETWRKIF[NR,OR]
```

The linker generates the NETWRKIF.RSP file.

If RMAC and LINK are not available, then you must use ASM, PIP, and GENMOD as shown below:

```
A>ASM NETWRKIF
    ;assemble with ORG 0000H
A>REN NTWRK0.HEX=NETWRKIF.HEX
    ;edit NETWRKIF.ASM ORG statement
A>ASM NETWRKIF
    ;assemble with ORG 0100H
A>REN NTWRK1.HEX=NETWRKIF.HEX
A>PIP NETWRKIF.HEX=NTWRK0.HEX,NTWRK1.HEX
    ;concatenate the HEX files
A>GENMOD NETWRKIF.HEX NETWRKIF.RSP
    ;generate the NETWRKIF RSP file
```

5) Copy the following files to the server.

```
SLVSP.RSP     = Slave Support Process
BROADCST.PRL = Broadcast mail
MRCVMAIL.PRL = Master receive mail
MSNDMAIL.PRL = Master send mail
NETWRKIF.RSP = Custom Network Interface Process
```

6) Perform a GENSYS on the MP/M system. The GENSYS must include the SLVSP.RSP file, the customized NETWRKIF.RSP, and can include the SPOOL.RSP.

When GENSYS asks for the number of consoles, do not include the consoles (character I/O drivers) that support the requesters. Generally, the response is 1.

7) Before beginning to debug CP/NET, verify that data can be sent and received between the server and the requester. Check to see that all 8 bits are transmitted and received if configured for 8-bit operation.

The sample NETWRKIF contains a debug conditional assembly flag that permits generation of a NETWRKIF.COM file. The NETWRKIF.COM version can debug a single requester as follows:

- Perform a GENSYS in which the SLVSP.RSP is included but not a NETWRKIF.RSP. During the GENSYS, do not specify bank switched memory.

- Execute the MPM.SYS produced from GENSYS and load the NETWRKIF.COM file with DDT .

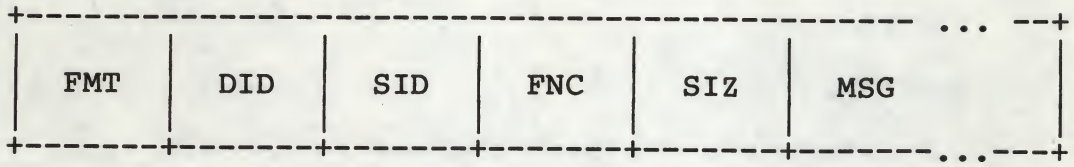
- Use DDT to debug the NETWRKIF process. This works only for a single requester.

There is, of course, much more that could be written about debugging CP/NET. However, NETWRKIF.ASM and SNIOS.ASM provide a very simple vehicle for porting CP/NET to other hardware. It has been our experience that porting CP/NET can require changes to as little as 50-100 lines of code.

The most challenging aspects of debugging CP/NET occur when extending beyond a simple portable version of the NETWRKIF and SNIOS to more elaborate protocols and physical methods of connecting a network. In fact, the first recommended change is to replace the character I/O interface used by the sample NETWRKIF with code that buffers an entire network logical message at the interrupt level and then sets a flag only when the entire message is received. This technique significantly improves performance because only one dispatch is produced per message rather than one per byte of the received message.

End of Section 4

Appendix A
CP/NET 1.1 Standard Message Formats



FMT = Message format code
DID = Message destination processor ID
SID = Message source processor ID
FNC = MP/M function code
SIZ = Data field length - 1
MSG = Actual message, SIZ + 1 bytes long

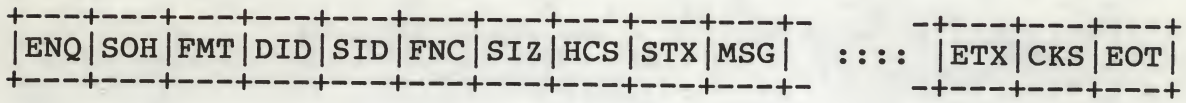
Message Field Length Table

FMT CODE	FMT	DID	SID	FNC	SIZ	MSG	Comment
00	1	1	1	1	1	1-256	Preferred format
01	1	1	1	1	1	1-256	Returned result
02	1	1	1	1	2	1-65536	
03	1	1	1	1	2	1-65536	Returned result
04	1	2	2	1	1	1-256	
05	1	2	2	1	1	1-256	Returned result
06	1	2	2	1	2	1-65536	
07	1	2	2	1	2	1-65536	Returned result

Appendix B
Recommended Server-Requester Handshake for RS-232C

Source		Destination	Comment
5 - ENQ	---------->		
	<----------	ACK - 6	
1 - SOH	---------->		
FMT	---------->		
DID	---------->		
SID	---------->		
FNC	---------->		
SIZ	---------->		
HCS	---------->		Modulo 256 sum from SOH to HCS = 0
	<----------	ACK - 6	
2 - STX	---------->		
DB0	---------->		First data byte
DBn	---------->		
3 - ETX	---------->		
CKS	---------->		Modulo 256 sum from STX to CKS = 0
4 - EOT	---------->		
	<----------	ACK - 6	

Appendix C
Recommended RS-232C 8-bit Network Protocol



Message format codes 00 & 01 are recommended.

Field Description:

ENQ = Enquire, one byte, 05H

SOH = Start of Header, one byte, 01H

FMT,DID,SID,FNC,SIZ = as defined in Appendix A, one byte per field

HCS = Header Checksum,one byte. This is a simple horizontal checksum. It is computed by adding together all the bytes of the message starting with the SOH to the SIZ byte of the header field modulo 256,complementing the result and adding one. Thus, adding together the entire message from the SOH to and including the HCS should give a total of zero.

STX = Start of Data, one byte, 02H

MSG = SIZ + 1 bytes long

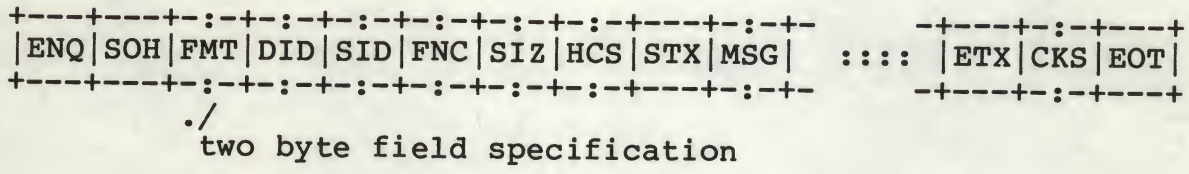
ETX = End of Data, one byte, 03H

CKS = Checksum, one byte. This is a simple horizontal checksum. It is computed by adding together all the bytes of the message starting with the STX to the last byte of the MSG field modulo 256, complementing the result and adding one. Thus, adding together the entire message from the STX to and including the CKS should give a total of zero.

EOT = End of Transmission, one byte, 04H

Appendix D
Recommended RS-232C 7-bit ASCII Network Protocol

Note: The 7-bit ASCII network protocol is identical to the 8-bit protocol with the exception that it requires twice as many bytes because each byte is transmitted in hexadecimal ASCII format.



Message format codes 00 & 01 are recommended.

Field Description:

ENQ = Enquire, one byte, 05H

SOH = Start of Header, one byte, 01H

FMT,DID,SID,FNC,SIZ = as defined in Appendix A, two bytes per field

HCS = Header Checksum,2 bytes (Hex-ASCII). This is a simple horizontal checksum. It is computed by adding together all the bytes of the message starting with the SOH to the SIZ of the header field modulo 256, complementing the result and adding one. Thus, adding together the entire message from the SOH to and including the HCS should give a total of zero.

STX = Start of Data, one byte, 02H

MSG = 2 * (SIZ + 1) bytes long

ETX = End of Data, one byte, 03H

CKS = Checksum, two bytes (Hex-ASCII). This is a simple horizontal checksum. It is computed by adding together all the bytes of the message starting with the FMT to the last byte of the MSG field modulo 256, complementing the result and adding one. Thus, adding together the entire message from the FMT to and including the CKS should give a total of zero.

EOT = End of Transmission, one byte, 04H

Appendix E
CP/NET 1.1 Logical Message Specification

Notes: ss = Server ID
rr = Requestor ID
xx = Don't care byte
nn = Value specified

All numeric values are in Hexadecimal.

FMT	DID	SID	FNC	SIZ	MSG / Function Name
					SYSTEM RESET: * NOT IMPLEMENTED AT SERVER *
00	ss	rr	00	00	00-00 = xx
01	rr	ss	00	00	00-00 = 00
					CONSOLE INPUT: * NOT IMPLEMENTED AT SERVER *
00	ss	rr	01	00	00-00 = xx
01	rr	ss	01	00	00-00 = 00
					CONSOLE OUTPUT: * NOT IMPLEMENTED AT SERVER *
00	ss	rr	02	00	00-00 = xx
01	rr	ss	02	00	00-00 = 00
					RAW CONSOLE INPUT:
00	ss	rr	03	00	00-00 = Server Console #
01	rr	ss	03	00	00-00 = Character Input
					RAW CONSOLE OUTPUT:
00	ss	rr	04	01	00-00 = Server Console # 01-01 = Character to Output
01	rr	ss	04	00	00-00 = 00

FMT	DID	SID	FNC	SIZ	MSG / Function Name
					LIST OUTPUT:
00	ss	rr	05	nn	00-00 = Server List # 01-nn = Characters to List Device (nn = 01 to 80)
01	rr	ss	05	00	00-00 = 00
					DIRECT CONSOLE I/O: * NOT IMPLEMENTED AT SERVER *
00	ss	rr	06	00	00-00 = xx
01	rr	ss	06	00	00-00 = 00
					GET I/O BYTE: * NOT IMPLEMENTED AT SERVER *
00	ss	rr	07	00	00-00 = xx
01	rr	ss	07	00	00-00 = 00
					SET I/O BYTE: * NOT IMPLEMENTED AT SERVER *
00	ss	rr	08	00	00-00 = xx
01	rr	ss	08	00	00-00 = 00
					PRINT STRING: * NOT IMPLEMENTED AT SERVER *
00	ss	rr	09	00	00-00 = xx
01	rr	ss	09	00	00-00 = 00
					READ CONSOLE BUFFER: * NOT IMPLEMENTED AT SERVER *
00	ss	rr	0A	00	00-00 = xx
01	rr	ss	0A	00	00-00 = 00
					GET CONSOLE STATUS:
00	ss	rr	0B	00	00-00 = Server Console #
01	rr	ss	0B	00	00-00 = Console Status Byte

FMT	DID	SID	FNC	SIZ	MSG / Function Name
					RETURN VERSION NUMBER: * NOT IMPLEMENTED AT SERVER *
00	ss	rr	0C	00	00-00 = xx
01	rr	ss	0C	00	00-00 = 00
					RESET DISK SYSTEM: * NOT IMPLEMENTED AT SERVER *
00	ss	rr	0D	00	00-00 = xx
01	rr	ss	0D	00	00-00 = 00
					SELECT DISK:
00	ss	rr	0E	00	00-00 = Selected Disk
01	rr	ss	0E	00	00-00 = Return Code
					OPEN FILE:
00	ss	rr	0F	0D	00-00 = User Number 01-01 = Drive Code 02-09 = File Name 0A-0C = File Type 0D-0D = Extent Number
01	rr	ss	0F	03	00-01 = FCB Address in Server 02-02 = Record Count 03-03 = Directory Code
					CLOSE FILE:
00	ss	rr	10	02	00-00 = User Number 01-02 = FCB Address in Server
01	rr	ss	10	00	00-00 = Directory Code

FMT	DID	SID	FNC	SIZ	MSG / Function Name
					SEARCH FOR FIRST:
00	ss	rr	11	0F	00-00 = User Number 01-01 = Drive Code 02-09 = File Name 0A-0C = File Type 0D-0D = Extent Number 0E-0E = S1 (not used) 0F-0F = S2
01	rr	ss	11	20	00-00 = Directory Code 01-20 = Directory FCB Entry
					SEARCH FOR NEXT:
00	ss	rr	12	00	00-00 = xx
01	rr	ss	12	20	00-00 = Directory Code 01-20 = Directory FCB Entry
					DELETE FILE:
00	ss	rr	13	0D	00-00 = User Number 01-01 = Drive Code 02-09 = File Name 0A-0C = File Type 0D-0D = Extent Number
01	rr	ss	13	00	00-00 = Directory Code
					READ SEQUENTIAL:
00	ss	rr	14	05	00-00 = User Number 01-02 = FCB Address in Server 03-03 = Extent Number 04-04 = Record Count 05-05 = Current Record
01	rr	ss	14	83	00-00 = Extent Number 01-01 = Record Count 02-02 = Current Record 03-82 = Sector of Data Read 83-83 = Return Code

FMT	DID	SID	FNC	SIZ	MSG / Function Name
					WRITE SEQUENTIAL:
00	ss	rr	15	85	00-00 = User Number 01-02 = FCB Address in Server 03-03 = Extent Number 04-04 = Record Count 05-05 = Current Record 06-85 = Sector of Data to Write
01	rr	ss	15	03	00-00 = Extent Number 01-01 = Record Count 02-02 = Current Record 03-03 = Return Code
					MAKE FILE:
00	ss	rr	16	0D	00-00 = User Number 01-01 = Drive Code 02-09 = File Name 0A-0C = File Type 0D-0D = Extent Number
01	rr	ss	16	03	00-01 = FCB Address in Server 02-02 = Record Count 03-03 = Directory Code
					RENAME FILE:
00	ss	rr	17	20	00-00 = User Number 01-01 = Drive Code 02-09 = File Name 0A-0C = File Type 0D-0D = Extent Number 0E-0E = S1 (not used) 0F-0F = S2 (not used) 10-10 = Record Count (not used) 11-11 = Drive Code 12-19 = File Name 1A-1C = File Type 1D-1D = Extent Number 1E-1E = S1 (not used) 1F-1F = S2 (not used) 20-20 = Record Count (not used)
01	rr	ss	17	00	00-00 = Directory Code

FMT	DID	SID	FNC	SIZ	MSG / Function Name
					RETURN LOGIN VECTOR:
00	ss	rr	18	00	00-00 = xx
01	rr	ss	18	01	00-01 = Login Vector
					RETURN CURRENT DISK: * NOT IMPLEMENTED AT SERVER *
00	ss	rr	19	00	00-00 = xx
01	rr	ss	19	00	00-00 = 00
					SET DMA ADDRESS: * NOT IMPLEMENTED AT SERVER *
00	ss	rr	1A	00	00-00 = xx
01	rr	ss	1A	00	00-00 = 00
					GET ALLOCATION VECTOR ADDRESS:
00	ss	rr	1B	00	00-00 = Current Drive
01	rr	ss	1B	FF	00-FF = Allocation Vector
					WRITE PROTECT DISK:
00	ss	rr	1C	00	00-00 = Current Drive
01	rr	ss	1C	00	00-00 = 00
					GET R/O VECTOR:
00	ss	rr	1D	00	00-00 = xx
01	rr	ss	1D	01	00-01 = R/O Vector

FMT	DID	SID	FNC	SIZ	MSG / Function Name
					SET FILE ATTRIBUTES:
00	ss	rr	1E	0D	00-00 = User Number 01-01 = Drive Code 02-09 = File Name 0A-0C = File Type 0D-0D = Extent Number
01	rr	ss	1E	00	00-00 = Directory Code
					GET DISK PARAMETER ADDRESS:
00	ss	rr	1F	00	00-00 = Current Drive
01	rr	ss	1F	0F	00-0F = Disk Parameter Block
					SET/GET USER CODE: * NOT IMPLEMENTED AT SERVER *
00	ss	rr	20	00	00-00 = Set/Get Code
01	rr	ss	20	00	00-00 = Current Code (if Get)
					READ RANDOM:
00	ss	rr	21	05	00-00 = User Number 01-02 = FCB Address in Server 03-05 = R0,R1,R2 Random Record #
01	rr	ss	21	83	00-00 = Extent Number 01-01 = Record Count 02-02 = Current Record 03-82 = Sector of Data Read 83-83 = Return Code
					WRITE RANDOM:
00	ss	rr	22	85	00-00 = User Number 01-02 = FCB Address in Server 03-82 = Sector of Data to Write 83-85 = R0,R1,R2 Random Record #
01	rr	ss	22	03	00-00 = Extent Number 01-01 = Record Count 02-02 = Current Record 03-03 = Return Code

FMT	DID	SID	FNC	SIZ	MSG / Function Name
					COMPUTE FILE SIZE:
00	ss	rr	23	0D	00-00 = User Number 01-01 = Drive Code 02-09 = File Name 0A-0C = File Type 0D-0D = Extent Number
01	rr	ss	23	03	00-02 = R0,R1,R2 Random Record # 03-03 = Return Code
					SET RANDOM RECORD:
00	ss	rr	24	05	00-00 = User Number 01-02 = FCB Address in Server 03-03 = Extent Number 04-04 = Record Count 05-05 = Current Record
01	rr	ss	24	03	00-02 = R0,R1,R2 Random Record # 03-03 = 00
					RESET DRIVE:
00	ss	rr	25	01	00-01 = Drive Vector
01	rr	ss	25	00	00-00 = Return Code
					ACCESS DRIVE:
00	ss	rr	26	01	00-01 = Drive Vector
01	rr	ss	26	00	00-00 = 00
					FREE DRIVE:
00	ss	rr	27	01	00-01 = Drive Vector
01	rr	ss	27	00	00-00 = 00

FMT	DID	SID	FNC	SIZ	MSG / Function Name
					WRITE RANDOM WITH ZERO FILL:
00	ss	rr	28	85	00-00 = User Number 01-02 = FCB Address in Server 03-82 = Sector of Data to Write 83-85 = R0,R1,R2 Random Record #
01	rr	ss	28	03	00-00 = Extent Number 01-01 = Record Count 02-02 = Current Record 03-03 = Return Code

CP/NET 1.1 FUNCTIONS

FMT	DID	SID	FNC	SIZ	MSG / Function Name
					LOGIN:
00	ss	rr	40	07	00-07 = Password, 8 ASCII Chars
01	rr	ss	40	00	00-00 = Return Code
					LOGOFF:
00	ss	rr	41	00	00-00 = xx
01	rr	ss	41	00	00-00 = Return Code
					SEND MESSAGE ON NETWORK:
00	ss	rr	42	nn	00-00 = Message Destination ID 01-nn = Message
01	rr	ss	42	00	00-00 = Return Code
					RECEIVE MESSAGE FROM NETWORK:
00	ss	rr	43	00	00-00 = xx
01	rr	ss	43	nn	00-00 = Message Source ID 01-nn = Message
					GET NETWORK STATUS: * NOT IMPLEMENTED AT SERVER *
00	ss	rr	44	00	00-00 = xx
01	rr	ss	44	00	00-00 = 00
					GET CONFIGURATION TABLE ADDRESS: * NOT IMPLEMENTED AT SERVER *
00	ss	rr	45	00	00-00 = xx
01	rr	ss	45	00	00-00 = 00

Appendix F
NDOS Function Summary

FUNC	FUNCTION NAME	INPUT PARAMETERS	OUTPUT RESULTS
64	Login	see def	A = Err Code
65	Logoff	see def	none
66	Send Message on Ntwrk	DE = Message Adr	A = Err Code
67	Receive Msg from Ntwk	DE = Message Adr	A = Err Code
68	Get Network Status	none	A = Status byte
69	Get Config Table Adr	none	HL= Table Adr

```
                rcvmsgerrmsg:
0038 496C6C6567         db      'Illegal RCVMAIL command.'
0050 24                 db      '$'

                rcvmsgfailedmsg:
0051 4D61696C62         db      'Mailbox empty or not logged in to master.'
007A 24                 db      '$'

                HexASCIItbl:
007B 3031323334         db      '0123456789ABCDEF'
                        endif

                        page

                        DSEG


                ;       Slave Configuration Table
                configtbl:

                Network$status:
0000                    ds      1               ; network status byte
0001                    ds      1               ; slave processor ID number
0002                    ds      2               ; A:  Disk device
0004                    ds      2               ; B:   "
0006                    ds      2               ; C:   "
0008                    ds      2               ; D:   "
000A                    ds      2               ; E:   "
000C                    ds      2               ; F:   "
000E                    ds      2               ; G:   "
0010                    ds      2               ; H:   "
0012                    ds      2               ; I:   "
0014                    ds      2               ; J:   "
0016                    ds      2               ; K:   "
0018                    ds      2               ; L:   "
001A                    ds      2               ; M:   "
001C                    ds      2               ; N:   "
001E                    ds      2               ; O:   "
0020                    ds      2               ; P:   "

0022                    ds      2               ; console device

0024                    ds      2               ; list device:
0026                    ds      1               ;       buffer index
0027 00                 db      0               ;       FMT
0028 00                 db      0               ;       DID
0029 56                 db      Slave$ID        ;       SID (CP/NOS must still initialize)
002A 05                 db      5               ;       FNC
002B                    ds      1               ;       SIZ
002C                    ds      1               ;       MSG(0)  List number
002D                    ds      128             ;       MSG(1) ... MSG(128)

```

```
                msg$adr:
00AD                    ds      2               ; message address
                        if      modem
                timeout$retries equ 0           ; timeout a max of 256 times
                        else
0064 =          timeout$retries equ 100         ; timeout a max of 100 times
                        endif
000A =          max$retries equ 10              ; send message max of 10 times
                retry$count:
00AF                    ds      1

                Mailmsg:                        ; Mail box message area
00B0 00                 db      0               ; msg.fmt = 0
00B1 00                 db      $-$             ; msg.did
00B2 00                 db      $-$             ; msg.sid
00B3 00                 db      $-$             ; msg.fnc
00B4 00                 db      $-$             ; msg.siz
00B5                    ds      256             ; msg.msg(0) ... msg.msg(255)

                FirstPass:
01B5 FF                 db      0ffh

                ;       Network Status Byte Equates
                ;
0010 =          active          equ     0001$0000b      ; slave logged in on network
0002 =          rcverr          equ     0000$0010b      ; error in received message
0001 =          senderr         equ     0000$0001b      ; unable to send message

                ;       General Equates
                ;
0001 =          SOH     equ     01h             ; Start of Header
0002 =          STX     equ     02h             ; Start of Data
0003 =          ETX     equ     03h             ; End of Data
0004 =          EOT     equ     04h             ; End of Transmission
0005 =          ENQ     equ     05h             ; Enquire
0006 =          ACK     equ     06h             ; Acknowledge
000A =          LF      equ     0ah             ; Line Feed
000D =          CR      equ     0dh             ; Carriage Return
0015 =          NAK     equ     15h             ; Negative Acknowledge

0002 =          conout  equ     2               ; console output function
0009 =          print   equ     9               ; print string function
0043 =          rcvmsg  equ     67              ; receive message NDOS function
0040 =          login   equ     64              ; Login NDOS function

                ;       I/O Equates
                ;
                        if      DB82
                stati   equ     83h
                mski    equ     08h
                dprti   equ     80h
```

```
                        stato   equ     83h
                        msko    equ     10h
                        statc   equ     81h
                        mskc    equ     20h
                        dprto   equ     86h
                                endif

                                if      DSC2
                                if      modem
                        stati   equ     59h
                        mski    equ     02h
                        dprti   equ     58h

                        stato   equ     59h
                        msko    equ     01h
                        dprto   equ     58h
                                else
                        stati   equ     51h
                        mski    equ     02h
                        dprti   equ     50h

                        stato   equ     51h
                        msko    equ     01h
                        dprto   equ     50h
                                endif
                                endif

                                if      Altos
001F =                  stati   equ     1fh
0001 =                  mski    equ     01h
001E =                  dprti   equ     1eh

001F =                  stato   equ     1fh
0004 =                  msko    equ     04h
001E =                  dprto   equ     1eh
                                endif


                                page
```

```
252
253                             CSEG
254                     ;       Utility Procedures
255                     ;
256                     delay:                          ; delay for c[a] * 0.5 milliseconds
257   008B 3E06                 mvi     a,6
258                     delay1:
259   008D 0E86                 mvi     c,86h
260                     delay2:
261   008F 0D                   dcr     c
262   0090 C28F00               jnz     delay2
263   0093 3D                   dcr     a
264   0094 C28D00               jnz     delay1
265   0097 C9                   ret
266
267                             if      ASCII
268                     Nib$out:                        ; A = nibble to be transmitted in ASCII
269                             cpi     10
270                             jnc     nibAtoF         ; jump if A-F
271                             adi     '0'
272                             mov     c,a
273                             jmp     Char$out
274                     nibAtoF:
275                             adi     'A'-10
276                             mov     c,a
277                             jmp     Char$out
278                             endif
279
280                     Pre$Char$out:
281   0098 7A                   mov     a,d
282   0099 81                   add     c
283   009A 57                   mov     d,a             ; update the checksum in D
284
285                     nChar$out:                      ; C = byte to be transmitted
286                             if      Altos
287   009B 3E10                 mvi     a,10h
288   009D D31F                 out     stato
289                             endif
290   009F DB1F                 in      stato
291   00A1 E604                 ani     mskO
292   00A3 CA9B00               jz      nChar$out
293
294                             if      DB82
295                             in      statc
296                             ani     mskc
297                             jz      nChar$out
298                             endif
299
```

Appendix G
Slave Network I/O System

```
                        title   'Slave Network I/O System for CP/NET 1.1'
                        page    60

                    ;*****************************************************************
                    ;*****************************************************************
                    ;**                                                             **
                    ;**     S l a v e   N e t w o r k   I / O   S y s t e m         **
                    ;**                                                             **
                    ;*****************************************************************
                    ;*****************************************************************

                    ;/*
                    ;  Copyright (C) 1980, 1981, 1982
                    ;  Digital Research
                    ;  P.O. Box 579
                    ;  Pacific Grove, CA 93950
                    ;
                    ;  Revised:  April 24, 1982
                    ;*/

0000 =              false   equ     0
FFFF =              true    equu    not false

0000 =              cpnos   equ     false           ; cp/net system

0000 =              DSC2    equ     false
0000 =              DB82    equ     false
FFFF =              Altos   equ     true

FFFF =              always$retry    equ     true    ; force continuous retries

0000 =              modem   equ     false

0000 =              ASCII   equ     false

0000 =              debug   equ     false

                            CSEG
                            if      cpnos
                            extrn   BDOS
                            else
0005 =              BDOS    equ     0005h
                            endif

                    NIOS:
                            public  NIOS
                    ;       Jump vector for SNIOS entry points
0000 C30501                 jmp     ntwrkinit       ; network initialization
0003 C31401                 jmp     ntwrksts        ; network status
0006 C31F01                 jmp     cnfgtbladr      ; return config table addr
0009 C32301                 jmp     sendmsg         ; send message on network
```

```
000C C38F01          jmp     receivemsg      ; receive message from network
000F C33902          jmp     ntwrkerror      ; network error
0012 C33A02          jmp     ntwrkwboot      ; network warm boot

                     if      DB82
                slave$ID     equ     12h     ; slave processor ID number
                     endif
                     if      DSC2
                slave$ID     equ     34h
                     endif
                     if      Altos
0056 =          slave$ID     equ     56h
                     endif

                     if      cpnos
                ;    Initial Slave Configuration Table
                Initconfigtbl:
                     db      0000$0000b      ; network status byte
                     db      slave$ID        ; slave processor ID number
                     db      84h,0           ; A:  Disk device
                     db      81h,0           ; B:   "
                     db      82h,0           ; C:   "
                     db      83h,0           ; D:   "
                     db      80h,0           ; E:   "
                     db      85h,0           ; F:   "
                     db      86h,0           ; G:   "
                     db      87h,0           ; H:   "
                     db      88h,0           ; I:   "
                     db      89h,0           ; J:   "
                     db      8ah,0           ; K:   "
                     db      8bh,0           ; L:   "
                     db      8ch,0           ; M:   "
                     db      8dh,0           ; N:   "
                     db      8eh,0           ; O:   "
                     db      8fh,0           ; P:   "
                     db      0,0             ; console device
                     db      0,0             ; list device:
                     db      0               ;       buffer index
                     db      0               ;       FMT
                     db      0               ;       DID
                     db      slave$ID        ;       SID
                     db      5               ;       FNC
                initcfglen equ  $-initconfigtbl
                     endif

                     if      not cpnos       ; see ntwrkwboot routine for why this
0000 =          defaultmaster   equ     00h  ;   won't go in cp/nos

                wboot$msg:                   ; data for warm boot routine
0015 3C5761726D      db      '<Warm Boot>'
0020 24              db      '$'

                networkerrmsg:
0021 4E6574776F      db      'Network access failed.'
0037 24              db      '$'
```

```
                        if      DSC2
                        nop                     ; these NOP's make DB8/2 & DSC2
                        nop                     ;  versions the same length - saves
                        nop                     ;  a second listing
                        nop
                        nop
                        nop
                        nop
                        endif

00A6 79                 mov     a,c
00A7 D31E               out     dprta
00A9 C9                 ret
                ;
                Char$out:
00AA CD9B00             call    nChar$out
                        if      Altos
00AD E3E3E3E3           xthl! xthl! xthl! xthl
00B1 E3E3E3E3           xthl! xthl! xthl! xthl
00B5 E3E3E3E3           xthl! xthl! xthl! xthl  ;delay 54 usec
00B9 C9                 ret
                        else
                        jmp     delay           ; delay after each Char sent to Mstr
                ;       ret
                        endif

                        if      ASCII
                Nib$in:                         ; return nibble in A register
                        call    Char$in
                        rc
                        ani     7fh
                        sui     '0'
                        cpi     10
                        jc      Nib$in$rtn      ; must be 0-9
                        adi     ('0'-'A'+10) and 0ffh
                        cpi     16
                        jc      Nib$in$rtn      ; must be 10-15
                        lda     network$status
                        ori     rcverr
                        sta     network$status
                        mvi     a,0
                        stc                     ; carry set indicating err cond
                        ret

                Nib$in$rtn:
                        ora     a               ; clear carry & return
                        ret
                        endif

                xChar$in:
00BA 0664               mvi     b,100           ; 100 ms  corresponds to longest possible
00BC C3C100             jmp     char$in0        ;wait between master operations
```

```
352
353                     Char$in:                        ; return byte in A register
354                                                     ;  carry set on rtn if timeout
355                             if      modem
356                             mvi     b,0             ; 256 ms = 7.76 chars @ 300 baud
357                             else
358                             if      Altos
359   00BF 0603                 mvi     b,3             ; 3 ms = 50 chars @ 125K baud
360                             else
361                             mvi     b,50            ; 50 ms = 50 chars @ 9600 baud
362                             endif
363                             endif
364                     Char$in0:
365   00C1 0E5A                 mvi     c,5ah
366                     Char$in1:
367                             if      Altos
368   00C3 3E00                 mvi     a,0
369   00C5 D31F                 out     stati
370                             endif
371   00C7 DB1F                 in      stati
372   00C9 E601                 ani     mskI
373   00CB C2D800               jnz     Char$in2
374   00CE 0D                   dcr     c
375   00CF C2C300               jnz     Char$in1
376   00D2 05                   dcr     b
377   00D3 C2C100               jnz     Char$in0
378   00D6 37                   stc                     ; carry set for err cond = timeout
379   00D7 C9                   ret
380                     Char$in2:
381   00D8 DB1E                 in      dprti
382   00DA C9                   ret                     ; rtn with raw char and carry cleared
383
384                     Net$out:                        ; C = byte to be transmitted
385                                                     ; D = checksum
386   00DB 7A                   mov     a,d
387   00DC 81                   add     c
388   00DD 57                   mov     d,a
389
390                             if      ASCII
391                             mov     a,c
392                             mov     b,a
393                             rar
394                             rar
395                             rar
396                             rar
397                             ani     0FH             ; mask HI-LO nibble to LO nibble
398                             call    Nib$out
399                             mov     a,b
400                             ani     0FH
401                             jmp     Nib$out
402
403                             else
404   00DE C3AA00               jmp     Char$out
405                             endif
```

```
406
407                     Msg$in:                         ; HL = destination address
408                                                     ; E  = # bytes to input
409   00E1 CDEC00               call    Net$in
410   00E4 D8                   rc
411   00E5 77                   mov     m,a
412   00E6 23                   inx     h
413   00E7 1D                   dcr     e
414   00E8 C2E100               jnz     Msg$in
415   00EB C9                   ret
416
417                     Net$in:                         ; byte returned in A register
418                                                     ; D  = checksum accumulator
419
420                               if      ASCII
421                               call    Nib$in
422                               rc
423                               add     a
424                               add     a
425                               add     a
426                               add     a
427                               push    psw
428                               call    Nib$in
429                               pop     b
430                               rc
431                               ora     b
432
433                               else
434   00EC CDBF00               call    Char$in         ;receive byte in Binary mode
435   00EF D8                   rc
436                               endif
437
438                     chks$in:
439   00F0 47                   mov     b,a
440   00F1 82                   add     d               ; add & update checksum accum.
441   00F2 57                   mov     d,a
442   00F3 B7                   ora     a               ; set cond code from checksum
443   00F4 78                   mov     a,b
444   00F5 C9                   ret
445
```

```
446                     Msg$out:                        ; HL = source address
447                                                     ; E  = # bytes to output
448                                                     ; D  = checksum
449                                                     ; C  = preamble byte
450  00F6 1600                  mvi     d,0             ; initialize the checksum
451  00F8 CD9800                call    Pre$Char$out    ; send the preamble character
452                     Msg$out$loop:
453  00FB 4E                    mov     c,m
454  00FC 23                    inx     h
455  00FD CDDB00                call    Net$out
456  0100 1D                    dcr     e
457  0101 C2FB00                jnz     Msg$out$loop
458  0104 C9                    ret
459
460                             page
461
462                     ;       Network Initialization
463                     ntwrkinit:
464
465                             if      cpnos           ; copy down network assignments
466                             lxi     h,Initconfigtbl
467                             lxi     d,configtbl
468                             mvi     c,initcfglen
469                     initloop:
470                             mov     a,m
471                             stax    d
472                             inx     h
473                             inx     d
474                             dcr     c
475                             jnz     initloop                ; initialize config tbl from ROM
476
477                             else
478  0105 3E56                  mvi     a,slave$ID              ;initialize slave ID byte
479  0107 320100                sta     configtbl+1             ;  in the configuration tablee
480                             endif
481
482                     ;       device initialization, as required
483
484                             if      Altos
485  010A 3E47                  mvi     a,047h
486  010C D30E                  out     0eh
487  010E 3E01                  mvi     a,1
488  0110 D30E                  out     0eh
489                             endif
490
491                             if      DSC2 and modem
492                             mvi     a,0ceh
493                             out     stato
494                             mvi     a,027h
495                             out     stato
496                             endif
```

```
497
498                             if      cpnos
499                             call    loginpr                 ; login to a master
500                             endif
501
502                     initok:
503   0112 AF                   xra     a                       ; return code is 0=success
504   0113 C9                   ret
505
506
507                             page
508
509                     ;       Network Status
510                     ntwrksts:
511   0114 3A0000               lda     network$status
512   0117 47                   mov     b,a
513   0118 E6FC                 ani     not (rcverr+senderr)
514   011A 320000               sta     network$status
515   011D 78                   mov     a,b
516   011E C9                   ret
517
518
519
520                     ;       Return Configuration Table Address
521                     cnfgtbladr:
522   011F 210000               lxi     h,configtbl
523   0122 C9                   ret
524
525
526                             page
```

```
527
528                     ;       Send Message on Network
529                     sendmsg:                        ; BC = message addr
530     0123 60                 mov     h,b
531     0124 69                 mov     l,c             ; HL = message address
532     0125 22AD00             shld    msg$adr
533                     re$sendmsg:
534     0128 3E0A               mvi     a,max$retries
535     012A 32AF00             sta     retry$count     ; initialize retry count
536                     send:
537     012D 2AAD00             lhld    msg$adr
538     0130 0E05               mvi     c,ENQ
539     0132 CDAA00             call    Char$out        ; send ENQ to master
540     0135 1664               mvi     d,timeout$retries
541                     ENQ$response:
542     0137 CDBF00             call    Char$in
543     013A D24401             jnc     got$ENQ$response
544     013D 15                 dcr     d
545     013E C23701             jnz     ENQ$response
546     0141 C38701             jmp     Char$in$timeout
547                     got$ENQ$response:
548     0144 CD7A01             call    get$ACK0
549     0147 0E01               mvi     c,SOH
550     0149 1E05               mvi     e,5
551     014B CDF600             call    Msg$out         ; send SOH FMT DID SID FNC SIZ
552     014E AF                 xra     a
553     014F 92                 sub     d
554     0150 4F                 mov     c,a
555     0151 CDDB00             call    net$out         ; send HCS (header checksum)
556     0154 CD7401             call    get$ACK
557     0157 2B                 dcx     h
558     0158 5E                 mov     e,m
559     0159 23                 inx     h
560     015A 1C                 inr     e
561     015B 0E02               mvi     c,STX
562     015D CDF600             call    Msg$out         ; send STX DB0 DB1 ...
563     0160 0E03               mvi     c,ETX
564     0162 CD9800             call    Pre$Char$out    ; send ETX
565     0165 AF                 xra     a
566     0166 92                 sub     d
567     0167 4F                 mov     c,a
568     0168 CDDB00             call    Net$out         ; send the checksum
569     016B 0E04               mvi     c,EOT
570     016D CD9B00             call    nChar$out       ; send EOT
571     0170 CD7401             call    get$ACK         ; (leave these
572     0173 C9                 ret                     ;             two instructions)
573
574                     get$ACK:
575     0174 CDBF00             call    Char$in
576     0177 DA7F01             jc      send$retry      ; jump if timeout
577                     get$ACK0:
578     017A E67F               ani     7fh
579     017C D606               sui     ACK
580     017E C8                 rz
```

```
581                     send$retry:
582  017F E1                    pop     h               ; discard return address
583  0180 21AF00                lxi     h,retry$count
584  0183 35                    dcr     m
585  0184 C22D01                jnz     send            ; send again unles max retries
586                     Char$in$timeout:
587  0187 3E01                  mvi     a,senderr
588
589                             if      always$retry
590  0189 CD2E02                call    error$return
591  018C C32801                jmp     re$sendmsg
592                             else
593                             jmp     error$return
594                             endif
595
596                             page
597
598                     ;       Receive Message from Network
599                     receivemsg:                     ; BC = message addr
600  018F 60                    mov     h,b
601  0190 69                    mov     l,c             ; HL = message address
602  0191 22AD00                shld    msg$adr
603                     re$receivemsg:
604  0194 3E0A                  mvi     a,max$retries
605  0196 32AF00                sta     retry$count     ; initialize retry count
606                     re$call:
607  0199 CDAB01                call    receive         ; rtn from receive is receive error
608
609                     receive$retry:
610  019C 21AF00                lxi     h,retry$count
611  019F 35                    dcr     m
612  01A0 C29901                jnz     re$call
613                     receive$timeout:
614  01A3 3E02                  mvi     a,rcverr
615
616                             if      always$retry
617  01A5 CD2E02                call    error$return
618  01A8 C39401                jmp     re$receivemsg
619                             else
620                             jmp     error$return
621                             endif
622
```

```
623                     receive:
624   01AB 2AAD00               lhld    msg$adr
625   01AE 1664                 mvi     d,timeout$retries
626                     receive$firstchar:
627   01B0 CDBA00               call    xcharin
628   01B3 D2BE01               jnc     got$firstchar
629   01B6 15                   dcr     d
630   01B7 C2B001               jnz     receive$firstchar
631   01BA E1                   pop     h               ; discard receive$retry rtn adr
632   01BB C3A301               jmp     receive$timeout
633                     got$firstchar:
634   01BE E67F                 ani     7fh
635   01C0 FE05                 cpi     ENQ             ; Enquire?
636   01C2 C2AB01               jnz     receive
637
638   01C5 0E06                 mvi     c,ACK
639   01C7 CD9B00               call    nChar$out       ; acknowledge ENQ with an ACK
640
641   01CA CDBF00               call    Char$in
642   01CD D8                   rc                      ; return to receive$retry
643   01CE E67F                 ani     7fh
644   01D0 FE01                 cpi     SOH             ; Start of Header ?
645   01D2 C0                   rnz                     ; return to receive$retry
646   01D3 57                   mov     d,a             ; initialize the HCS
647   01D4 1E05                 mvi     e,5
648   01D6 CDE100               call    Msg$in
649   01D9 D8                   rc                      ; return to receive$retry
650   01DA CDEC00               call    Net$in
651   01DD D8                   rc                      ; return to receive$retry
652   01DE C22902               jnz     bad$checksum
653   01E1 CD2102               call    send$ACK
654   01E4 CDBF00               call    Char$in
655   01E7 D8                   rc                      ; return to receive$retry
656   01E8 E67F                 ani     7fh
657   01EA FE02                 cpi     STX             ; Start of Data ?
658   01EC C0                   rnz                     ; return to receive$retry
659   01ED 57                   mov     d,a             ; initialize the CKS
660   01EE 2B                   dcx     h
661   01EF 5E                   mov     e,m
662   01F0 23                   inx     h
663   01F1 1C                   inr     e
664   01F2 CDE100               call    msg$in          ; get DB0 DB1 ...
665   01F5 D8                   rc                      ; return to receive$retry
666   01F6 CDBF00               call    Char$in         ; get the ETX
667   01F9 D8                   rc                      ; return to receive$retry
668   01FA E67F                 ani     7fh
669   01FC FE03                 cpi     ETX
```

```
670  01FE C0            rnz                    ; return to receive$retry
671  01FF 82            add     d
672  0200 57            mov     d,a            ; update CKS with ETX
673  0201 CDEC00        call    Net$in         ; get CKS
674  0204 D8            rc                     ; return to receive$retry
675  0205 CDBF00        call    Char$in        ; get EOT
676  0208 D8            rc                     ; return to receive$retry
677  0209 E67F          ani     7fh
678  020B FE04          cpi     EOT
679  020D C0            rnz                    ; return to receive$retry
680  020E 7A            mov     a,d
681  020F B7            ora     a              ; test CKS
682  0210 C22902        jnz     bad$checksum
683  0213 E1            pop     h              ; discard receive$retry rtn adr
684  0214 2AAD00        lhld    msg$adr
685  0217 23            inx     h
686  0218 3A0100        lda     configtbl+1
687  021B 96            sub     m
688  021C CA2102        jz      send$ACK       ; jump with A=0 if DID ok
689  021F 3EFF          mvi     a,0ffh         ; return code shows bad DID
690                send$ACK:
691  0221 F5            push    psw            ; save return code
692  0222 0E06          mvi     c,ACK
693  0224 CD9B00        call    nChar$out      ; send ACK if checksum ok
694  0227 F1            pop     psw            ; restore return code
695  0228 C9            ret
696
697                bad$DID:
698                bad$checksum:
699  0229 0E15          mvi     c,NAK
700  022B C3AA00        jmp     Char$out       ; send NAK on bad chksm & not max retries
701                ;       ret
702
703                error$return:
704  022E 210000        lxi     h,network$status
705  0231 B6            ora     m
706  0232 77            mov     m,a
707  0233 CD3902        call    ntwrkerror     ; perform any required device re-init.
708  0236 3EFF          mvi     a,0ffh
709  0238 C9            ret
710
711                ntwrkerror:
712                                           ;  perform any required device
713  0239 C9            ret                    ;     re-initialization
714
715                     page
```

```
716
717                     ;
718                     ntwrkwboot:
719
720                     ;       This procedure is called each time the CCP is
721                     ;       reloaded from disk.  This version prints '<WARM BOOT>'
722                     ;       on the console and then checks for mail at the master,
723                     ;       but anything necessary for restart can be put here.
724
725                             if      not cpnos
726
727                     ;       NOTE:  The following code will not fit in a
728                     ;       4K ROM.  Consequently, we do not include it
729                     ;       in the CP/NOS version CPNIOS.ASM.  It would
730                     ;       fit in 5K if it needs to be included in CP/NOS.
731
732     023A 0E09               mvi     c,9
733     023C 111500             lxi     d,wboot$msg
734     023F CD0500             call    BDOS
735
736
737                     getmail:
738     0242 3AFF00             lda     0ffh
739     0245 32B501             sta     firstpass
740                     restart:
741     0248 3A0100             lda     configtbl+1     ; get slave ID
742     024B 32B200             sta     Mailmsg+2       ; Mailmsg.sid = configtbl.slaveID
743     024E 0E00               mvi     c, defaultmaster
744
745                     dorcvmsg:
746     0250 AF                 xra     a
747     0251 11B000             lxi     d, Mailmsg
748     0254 12                 stax    d               ; MSG.FMT = 0
749     0255 13                 inx     d
750     0256 79                 mov     a,c
751     0257 12                 stax    d               ; msg.did = [xx]
752     0258 13                 inx     d
753     0259 13                 inx     d
```

```
754   025A 3E43          mvi   a,rcvmsg
755   025C 12            stax  d             ; msg,fnc = rcvmsg
756   025D 13            inx   d
757   025E AF            xra   a
758   025F 12            stax  d
759   0260 01B000        lxi   b,Mailmsg
760   0263 CD2301        call  sendmsg       ; send message to network
761   0266 3C            inr   a
762   0267 CAE702        jz    networkerr
763   026A 01B000        lxi   b,Mailmsg
764   026D CD8F01        call  receivemsg    ; receive message from network
765   0270 3C            inr   a
766   0271 CAE702        jz    networkerr
767   0274 3AB400        lda   Mailmsg+4
768   0277 B7            ora   a
769   0278 C28202        jnz   displaymsg
770   027B 3AB500        lda   Mailmsg+5
771   027E 3C            inr   a
772   027F CADA02        jz    rcvmsgfailed
773                displaymsg:
774   0282 21B501        lxi   h,FirstPass
775   0285 34            inr   m
776   0286 CA9702        jz    noCRLF
777   0289 0E02          mvi   c,conout
778   028B 1E0D          mvi   e,CR
779   028D CD0500        call  BDOS
780   0290 0E02          mvi   c,conout
781   0292 1E0A          mvi   e,LF
782   0294 CD0500        call  BDOS
783                noCRLF:
784   0297 21B000        lxi   h,Mailmsg
785   029A 365B          mvi   m,'['
786   029C 23            inx   h
787   029D EB            xchg
788   029E 3AB500        lda   Mailmsg+5
789   02A1 217B00        lxi   h,HexASCIItbl
790   02A4 F5            push  psw
791   02A5 E5            push  h
792   02A6 E6F0          ani   0f0h
793   02A8 0F            rrc
794   02A9 0F            rrc
795   02AA 0F            rrc
796   02AB 0F            rrc
797   02AC 4F            mov   c,a
798   02AD 0600          mvi   b,0
799   02AF 09            dad   b
800   02B0 7E            mov   a,m
801   02B1 12            stax  d
802   02B2 13            inx   d
803   02B3 E1            pop   h
804   02B4 F1            pop   psw
805   02B5 E60F          ani   0fh
```

```
806   02B7 4F            mov     c,a
807   02B8 09            dad     b
808   02B9 7E            mov     a,m
809   02BA 12            stax    d
810   02BB EB            xchg
811   02BC 23            inx     h
812   02BD 365D          mvi     m,']'
813   02BF 23            inx     h
814   02C0 5E            mov     e,m
815   02C1 3620          mvi     m,' '
816   02C3 23            inx     h
817   02C4 3622          mvi     m,'"'
818   02C6 23            inx     h
819   02C7 1600          mvi     d,0
820   02C9 19            dad     d
821   02CA 3622          mvi     m,'"'
822   02CC 23            inx     h
823   02CD 3624          mvi     m,'$'
824   02CF 0E09          mvi     c,print
825   02D1 11B000        lxi     d,Mailmsg
826   02D4 CD0500        call    BDOS
827   02D7 C34802        jmp     restart
828
829                rcvmsgfailed:
830   02DA 3AB501        lda     FirstPass
831   02DD 3C            inr     a
832   02DE C2F502        jnz     Exit
833   02E1 115100        lxi     d,rcvmsgfailedmsg
834   02E4 C3F002        jmp     prntmsg
835
836                networkerr:
837   02E7 112100        lxi     d,networkerrmsg
838   02EA C3F002        jmp     prntmsg
839
840                rcvmsgerr:
841   02ED 113800        lxi     d,rcvmsgerrmsg
842                prntmsg:
843   02F0 0E09          mvi     c,print
844   02F2 CD0500        call    BDOS
845                ;       jmp     Exit
846                        endif
847
848                Exit:
849   02F5 C9            ret
850
851                        page
```

```
852
853                     if      cpnos
854             ;
855             ;       LOGIN to a Master
856             ;
857             ; Equates
858             ;
859             buff    equ     0080h
860
861             readbf  equ     10
862
863             active  equ     0001$0000b
864
865             loginpr:
866                     mvi     c,initpasswordmsglen
867                     lxi     h,initpasswordmsg
868                     lxi     d,passwordmsg
869             copypassword:
870                     mov     a,m
871                     stax    d
872                     inx     h
873                     inx     d
874                     dcr     c
875                     jnz     copypassword
876                     mvi     c,print
877                     lxi     d,loginmsg
878                     call    BDOS
879                     mvi     c,readbf
880                     lxi     d,buff-1
881                     mvi     a,50h
882                     stax    d
883                     call    BDOS
884                     lxi     h,buff
885                     mov     a,m     ; get # chars in the command tail
886                     ora     a
887                     jz      dologin ; default login if empty command tail
888                     mov     c,a     ; A = # chars in command tail
889                     xra     a
890                     mov     b,a     ; B will accumulate master ID
891             scanblnks:
892                     inx     h
893                     mov     a,m
894                     cpi     ' '
895                     jnz     postblnks ; skip past leading blanks
896                     dcr     c
897                     jnz     scanblnks
898                     jmp     prelogin ; jump if command tail exhausted
```

```
899             pastblnks:
900                     cpi     '['
901                     jz      scanMstrID
902                     mvi     a,8
903                     lxi     d,passwordmsg+5+8-1
904                     xchg
905             spacefill:
906                     mvi     m,' '
907                     dcx     h
908                     dcr     a
909                     jnz     spacefill
910                     xchg
911             scanLftBrkt:
912                     mov     a,m
913                     cpi     '['
914                     jz      scanMstrID
915                     inx     d
916                     stax    d       ;update the password
917                     inx     h
918                     dcr     c
919                     jnz     scanLftBrkt
920                     jmp     prelogin
921             scanMstrID:
922                     inx     h
923                     dcr     c
924                     jz      loginerr
925                     mov     a,m
926                     cpi     ']'
927                     jz      prelogin
928                     sui     '0'
929                     cpi     10
930                     jc      updateID
931                     adi     ('0'-'A'+10) and 0ffh
932                     cpi     16
933                     jnc     loginerr
934             updateID:
935                     push    psw
936                     mov     a,b
937                     add     a
938                     add     a
939                     add     a
940                     add     a
941                     mov     b,a     ; accum * 16
942                     pop     psw
943                     add     b
944                     mov     b,a
945                     jmp     scanMstrID
946
947             prelogin:
948                     mov     a,b
949
```

```
950             dologin:
951                     lxi     b,passwordmsg+1
952                     stax    b
953                     dcx     b
954                     call    sendmsg
955                     inr     a
956                     lxi     d,loginfailedmsg
957                     jz      printmsg
958                     lxi     b,passwordmsg
959                     call    receivemsg
960                     inr     a
961                     lxi     d,loginfailedmsg
962                     jz      printmsg
963                     lda     passwordmsg+5
964                     inr     a
965                     jnz     loginOK
966                     jmp     printmsg
967
968             loginerr:
969                     lxi     d,loginerrmsg
970             printmsg:
971                     mvi     c,print
972                     call    BDOS
973                     jmp     loginpr         ; try login again
974
975             loginOK:
976                     lxi     h,network$status ; HL = status byte addr
977                     mov     a,m
978                     ori     active  ; set active bit true
979                     mov     m,a
980                     ret
981
982             ;
983             ; Local Data Segment
984             ;
985             loginmsg:
986                     db      cr,lf
987                     db      'LOGIN = '
988                     db      '$'
989
990             initpasswordmsg:
991                     db      00h     ; FMT
992                     db      00h     ; DID Master ID #
993                     db      slave$ID ;SID
994                     db      40h     ; FNC
995                     db      7       ; SIZ
996                     db      'PASSWORD' ; password
997             initpasswordmsglen equ  $-initpasswordmsg
998
999
```

```
1000                    loginerrmsg:
1001                            db      lf
1002                            db      'Illegal LOGIN command.'
1003                            db      '$'
1004
1005                    loginfailedmsg:
1006                            db      lf
1007                            db      'LOGIN failed.'
1008                            db      '$'
1009
1010                            DSEG
1011                    passwordmsg:
1012                            ds      1       ; FMT
1013                            ds      1       ; DID
1014                            ds      1       ; SID
1015                            ds      1       ; FNC
1016                            ds      1       ; SIZ
1017                            ds      8       ; DAT = password
1018                            endif
1019
1020    02F6                    end
```

Appendix H
Master Network I/F Module

```
                        title   'Master Network I/F Module'
                        name    'netwrkif'
                        page    60
                    ;*****************************************************************
                    ;*****************************************************************
                    ;**                                                             **
                    ;**     M a s t e r   N e t w o r k   I / F   M o d u l e       **
                    ;**                                                             **
                    ;*****************************************************************
                    ;*****************************************************************

                    ;/*
                    ;  Copyright (C) 1980,1981,1982
                    ;  Digital Research
                    ;  P.O. Box 579
                    ;  Pacific Grove, CA 93950
                    ;
                    ;  Modified April 26, 1982
                    ;
                    ;*/


0000 =              false   equ     0
FFFF =              true    equ     not false

0000 =              z80     equu    false

0000 =              debug   equ     false
0000 =              modem   equ     false

FFFF =              mail    equ     true    ; allocate storage and setup mailbox

0000 =              WtchDg  equ     false   ; include watch dog timer

0000 =              mutexin equ     false   ; provide mutual exclusion on input
0000 =              mutexout equ    false   ; provide mutual exclusion on output

0010 =                      NmbFcbs equ     16      ; Number of fcbs for each slave

                            if      debug
                    NmbSlvs equ     1

                            lxi     sp,NtwrkIS0+2eh
                            mvi     c,145
                            mvi     e,64
                            call    bdos    ; set priority to 64
                            lxi     h,UQCBNtwrkQI0
                            lxi     d,UQCBNtwrkQ00
                            lxi     b,BufferQ0
                            mvi     a,00h
                            ret
--
```

```
                bdosadr:
                        dw      0005h

                        else
                ;NmbSlvs        equ     2
0001 =          NmbSlvs equ     1               ;BKS TEST
                bdosadr:
0000 0000               dw      $-$
                        endif

                ;  Network Interface Process #0
                ;
                NtwrkIP0:
0002 0000               dw      0               ; link
0004 00                 db      0               ; status
0005 40                 db      64              ; priority
0006 6400               dw      NtwrkIS0+46     ; stack pointer
0008 4E7477726B         db      'NtwrkIP0'      ; name
0010 00                 db      0               ; console
0011 FF                 db      0ffh            ; memseg
0012                    ds      2               ; b
0014                    ds      2               ; thread
0016                    ds      2               ; buff
0018                    ds      1               ; user code & disk slct
0019                    ds      2               ; dcnt
001B                    ds      1               ; searchl
001C                    ds      2               ; searcha
001E                    ds      2               ; active drives
0020 0000               dw      0               ; HL'
0022 0000               dw      0               ; DE'
0024 0000               dw      0               ; BC'
0026 0000               dw      0               ; AF'
0028 0000               dw      0               ; IY
002A 0000               dw      0               ; IX
002C 8000               dw      UQCBNtwrkQI0    ; HL
002E A000               dw      UQCBNtwrkQO0    ; DE
0030 A600               dw      BufferQ0        ; BC
0032 0000               dw      0               ; AF, A = ntwkif console dev #
0034                    ds      2               ; scratch

                NtwrkIS0:
0036 C7C7C7C7C7         dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
003E C7C7C7C7C7         dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
0046 C7C7C7C7C7         dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
004E C7C7C7C7C7         dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
0056 C7C7C7C7C7         dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
005E C7C7C7C7C7         dw      0c7c7h,0c7c7h,0c7c7h
0064 EF0A               dw      setup
```

```
                        QCBNtwrkQI0:
0066                    ds      2               ; link
0068 4E7477726B         db      'NtwrkQI0'      ; name
0070 0200               dw      2               ; msglen
0072 0100               dw      1               ; nmbmsgs
0074                    ds      2               ; dqph
0076                    ds      2               ; nqph
0078                    ds      2               ; msgin
007A                    ds      2               ; msgout
007C                    ds      2               ; msgcnt
007E                    ds      2               ; buffer

                        UQCBNtwrkQI0:
0080 6600               dw      QCBNtwrkQI0     ; pointer
0082 8400               dw      BufferQI0Addr   ; msgadr
                        BufferQI0Addr:
0084 A600               dw      BufferQ0

                        QCBNtwrkQ00:
0086                    ds      2               ; link
0088 4E7477726B         db      'NtwrkQ00'      ; name
0090 0200               dw      2               ; msglen
0092 0100               dw      1               ; nmbmsgs
0094                    ds      2               ; dqph
0096                    ds      2               ; nqph
0098                    ds      2               ; msgin
009A                    ds      2               ; msgout
009C                    ds      2               ; msgcnt
009E                    ds      2               ; buffer

                        UQCBNtwrkQ00:
00A0 8600               dw      QCBNtwrkQ00     ; pointer
00A2 A400               dw      Buffer00Addr    ; msgadr
                        Buffer00Addr:
00A4                    ds      2

                        Buffer00:
00A6                    ds      1               ; FMT
00A7                    ds      1               ; DID
00A8                    ds      1               ; SID
00A9                    ds      1               ; FNC
00AA                    ds      1               ; SIZ
00AB                    ds      256             ; MSG

                ;       Network Interface Process #1
                ;
                        if      NmbSlvs GE 2
                NtwrkIP1:
                        if      NmbSlvs GE 3
                        dw      NtwrkIP2        ; link
                        else
                        dw      0               ; link
                        endif
```

```
155                     db      0               ; status
156                     db      64              ; priority
157                     dw      NtwrkIS1+46     ; stack pointer
158                     db      'NtwrkIP1'      ; name
159                     db      0               ; console
160                     db      0ffh            ; memseg
161                     ds      2               ; b
162                     ds      2               ; thread
163                     ds      2               ; buff
164                     ds      1               ; user code & disk slct
165                     ds      2               ; dcnt
166                     ds      1               ; searchl
167                     ds      2               ; searcha
168                     ds      2               ; active drives
169                     dw      0               ; HL'
170                     dw      0               ; DE'
171                     dw      0               ; BC'
172                     dw      0               ; AF'
173                     dw      0               ; IY
174                     dw      0               ; IX
175                     dw      UQCBNtwrkQI1    ; HL
176                     dw      UQCBNtwrkQO1    ; DE
177                     dw      BufferQ1        ; BC
178                     dw      0100h           ; AF, A = ntwkif console dev #
179                     ds      2               ; scratch
180
181             NtwrkIS1:
182                     dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
183                     dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
184                     dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
185                     dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
186                     dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
187                     dw      0c7c7h,0c7c7h,0c7c7h
188                     dw      init
189
190             QCBNtwrkQI1:
191                     ds      2               ; link
192                     db      'NtwrkQI1'      ; name
193                     dw      2               ; msglen
194                     dw      1               ; nmbmsgs
195                     ds      2               ; dqph
196                     ds      2               ; nqph
197                     ds      2               ; msgin
198                     ds      2               ; msgout
199                     ds      2               ; msgcnt
200                     ds      2               ; buffer
201
202             UQCBNtwrkQI1:
203                     dw      QCBNtwrkQI1     ; pointer
204                     dw      BufferQI1Addr   ; msgadr
205             BufferQI1Addr:
206                     dw      BufferQ1
207
```

```
208             QCBNtwrkQO1:
209                     ds      2               ; link
210                     db      'NtwrkQO1'      ; name
211                     dw      2               ; msglen
212                     dw      1               ; nmbmsgs
213                     ds      2               ; dqph
214                     ds      2               ; nqph
215                     ds      2               ; msgin
216                     ds      2               ; msgout
217                     ds      2               ; msgcnt
218                     ds      2               ; buffer
219
220             UQCBNtwrkQO1:
221                     dw      QCBNtwrkQO1     ; pointer
222                     dw      BufferQO1Addr   ; msgadr
223             BufferQO1Addr:
224                     ds      2
225
226             BufferQ1:
227                     ds      1               ; FMT
228                     ds      1               ; DID
229                     ds      1               ; SID
230                     ds      1               ; FNC
231                     ds      1               ; SIZ
232                     ds      256             ; MSG
233                     endif
234
235             ;       Network Interface Process #2
236             ;
237                     if      NmbSlvs GE 3
238             NtwrkIP2:
239                     if      NmbSlvs GE 4
240                     dw      NtwrkIP3        ; link
241                     else
242                     dw      0               ; link
243                     endif
244                     db      0               ; status
245                     db      64              ; priority
246                     dw      NtwrkIS2+46     ; stack pointer
247                     db      'NtwrkIP2'      ; name
248                     db      0               ; console
249                     db      0ffh            ; memseg
250                     ds      2               ; b
251                     ds      2               ; thread
252                     ds      2               ; buff
253                     ds      1               ; user code & disk slct
254                     ds      2               ; dcnt
255                     ds      1               ; searchl
256                     ds      2               ; searcha
257                     ds      2               ; active drives
258                     dw      0               ; HL'
259                     dw      0               ; DE'
```

```
260                     dw      0               ; BC'
261                     dw      0               ; AF'
262                     dw      0               ; IY
263                     dw      0               ; IX
264                     dw      UQCBNtwrkQI2    ; HL
265                     dw      UQCBNtwrkQO2    ; DE
266                     dw      BufferQ2        ; BC
267                     dw      0200h           ; AF, A = ntwkif console dev #
268                     ds      2               ; scratch
269
270             NtwrkIS2:
271                     dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
272                     dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
273                     dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
274                     dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
275                     dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
276                     dw      0c7c7h,0c7c7h,0c7c7h
277                     dw      init
278
279             QCBNtwrkQI2:
280                     ds      2               ; link
281                     db      'NtwrkQI2'      ; name
282                     dw      2               ; msglen
283                     dw      1               ; nmbmsgs
284                     ds      2               ; dqph
285                     ds      2               ; nqph
286                     ds      2               ; msgin
287                     ds      2               ; msgout
288                     ds      2               ; msgcnt
289                     ds      2               ; buffer
290
291             UQCBNtwrkQI2:
292                     dw      QCBNtwrkQI2     ; pointer
293                     dw      BufferQI2Addr   ; msgadr
294             BufferQI2Addr:
295                     dw      BufferQ2
296
297             QCBNtwrkQO2:
298                     ds      2               ; link
299                     db      'NtwrkQO2'      ; name
300                     dw      2               ; msglen
301                     dw      1               ; nmbmsgs
302                     ds      2               ; dqph
303                     ds      2               ; nqph
304                     ds      2               ; msgin
305                     ds      2               ; msgout
306                     ds      2               ; msgcnt
307                     ds      2               ; buffer
308
309             UQCBNtwrkQO2:
310                     dw      QCBNtwrkQO2     ; pointer
311                     dw      BufferQO2Addr   ; msgadr
312             BufferQO2Addr:
313                     ds      2
```

```
314
315                 BufferQ2:
316                         ds      1               ; FMT
317                         ds      1               ; DID
318                         ds      1               ; SID
319                         ds      1               ; FNC
320                         ds      1               ; SIZ
321                         ds      256             ; MSG
322                         endif
323
324                 ;       Network Interface Process #3
325                 ;
326                         if      NmbSlvs GE 4
327                 NtwrkIP3:
328                         dw      0               ; link
329                         db      0               ; status
330                         db      64              ; priority
331                         dw      NtwrkIS3+46     ; stack pointer
332                         db      'NtwrkIP3'      ; name
333                         db      0               ; console
334                         db      0ffh            ; memseg
335                         ds      2               ; b
336                         ds      2               ; thread
337                         ds      2               ; buff
338                         ds      1               ; user code & disk slct
339                         ds      2               ; dcnt
340                         ds      1               ; searchl
341                         ds      2               ; searcha
342                         ds      2               ; active drives
343                         dw      0               ; HL'
344                         dw      0               ; DE'
345                         dw      0               ; BC'
346                         dw      0               ; AF'
347                         dw      0               ; IY
348                         dw      0               ; IX
349                         dw      UQCBNtwrkQI3    ; HL
350                         dw      UQCBNtwrkQO3    ; DE
351                         dw      BufferQ3        ; BC
352                         dw      0300h           ; AF, A = ntwkif console dev #
353                         ds      2               ; scratch
354
355                 NtwrkIS3:
356                         dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
357                         dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
358                         dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
359                         dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
360                         dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
361                         dw      0c7c7h,0c7c7h,0c7c7h
362                         dw      init
363
```

```
364             QCBNtwrkQI3:
365                     ds      2               ; link
366                     db      'NtwrkQI3'      ; name
367                     dw      2               ; msglen
368                     dw      1               ; nmbmsgs
369                     ds      2               ; dqph
370                     ds      2               ; nqph
371                     ds      2               ; msgin
372                     ds      2               ; msgout
373                     ds      2               ; msgcnt
374                     ds      2               ; buffer
375
376             UQCBNtwrkQI3:
377                     dw      QCBNtwrkQI3     ; pointer
378                     dw      BufferQI3Addr   ; msgadr
379             BufferQI3Addr:
380                     dw      BufferQ3
381
382             QCBNtwrkQO3:
383                     ds      2               ; link
384                     db      'NtwrkQO3'      ; name
385                     dw      2               ; msglen
386                     dw      1               ; nmbmsgs
387                     ds      2               ; dqph
388                     ds      2               ; nqph
389                     ds      2               ; msgin
390                     ds      2               ; msgout
391                     ds      2               ; msgcnt
392                     ds      2               ; buffer
393
394             UQCBNtwrkQO3:
395                     dw      QCBNtwrkQO3     ; pointer
396                     dw      BufferQO3Addr   ; msgadr
397             BufferQO3Addr:
398                     ds      2
399
400             BufferQ3:
401                     ds      1               ; FMT
402                     ds      1               ; DID
403                     ds      1               ; SID
404                     ds      1               ; FNC
405                     ds      1               ; SIZ
406                     ds      256             ; MSG
407                     endif
408
409
```

```
410                     if      WtchDg
411             ;  Watchdog Timer Process
412             ;
413             WatchDogPD:
414                     if      NmbSlvs GT 1
415                     dw      NtwrkIP1        ; link
416                     else
417                     dw      0               ; link
418                     endif
419                     db      0               ; status
420                     db      64              ; priority
421                     dw      WatchDogSTK+46  ; stack pointer
422                     db      'WatchDog'      ; name
423                     db      0               ; console
424                     db      0ffh            ; memseg
425                     ds      2               ; b
426                     ds      2               ; thread
427                     ds      2               ; buff
428                     ds      1               ; user code & disk slct
429                     ds      2               ; dcnt
430                     ds      1               ; searchl
431                     ds      2               ; searcha
432                     ds      2               ; active drives
433                     dw      0               ; HL'
434                     dw      0               ; DE'
435                     dw      0               ; BC'
436                     dw      0               ; AF'
437                     dw      0               ; IY
438                     dw      0               ; IX
439                     dw      0               ; HL
440                     dw      0               ; DE
441                     dw      0               ; BC
442                     dw      0               ; AF
443                     ds      2               ; scratch
444
445             WatchDogSTK:
446                     dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
447                     dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
448                     dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
449                     dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
450                     dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
451                     dw      0c7c7h,0c7c7h,0c7c7h
452                     dw      WatchDog
453
454             WatchDogTime:
455                     dw      $-$     ; one-second counter
456
457             WatchDogTable:
458                     ;       Waiting Timeout Start   Flag    Slave
459                     db      0,      0,      0,0,    0ah     ; #0
460                     db      0,      0,      0,0,    0bh     ; #1
461                     db      0,      0,      0,0,    0fh     ; #2
462                     db      0,      0,      0,0,    0dh     ; #3
463                     endif
```

```
464
465                     if      mutexin or mutexout
466                 QCBMXSXmitq:                    ; MX queue for slv xmiting
467                     ds      2                   ; link
468                     db      'MXSXmitq'          ; name
469                     dw      0                   ; msglen
470                     dw      1                   ; nmbmsgs
471                     ds      2                   ; dqph
472                     ds      2                   ; nqph
473                     ds      2                   ; msgin
474                     ds      2                   ; msgout
475                     ds      2                   ; msgcnt
476                     ds      2                   ; buffer (owner PD)
477
478                 UQCBMXSXmitq:
479                     dw      QCBMXSXmitq
480                 ;   dw      0
481                 ;   db      'MXSXmitq'
482                     endif
483
484                 ;   Master Configuration Table
485
486                 configtbl:
487 01AB 00             db      0       ; Master status byte
488 01AC 00             db      0       ; Master processor ID
489 01AD 01             db      NmbSlvs ; Maximum number of slaves supported
490 01AE 03             db      3       ; Number of logged in slaves
491 01AF 0E00           dw      000eh   ; 16 bit vector of logged in slaves
492 01B1 01             db      01
493 01B2 02             db      02
494 01B3 03             db      03
495 01B4                ds      13      ; Slave processor ID's
496 01C1 5041535357     db      'PASSWORD' ; login password
497 01C9 10             db      nmbfcbs         ; Number of FCB's for each slave
498 01CA E601           dw      fcbtable        ; Address of table containing fcbs
499
500 0001 =          nmsg    equ     1       ; number of messages buffered
501
502                 ;Slave0:                ; storage allocated in SLAVESP
503                 ;   ds      52      ; processor descriptor
504                 ;   ds      450     ; stack area for reentrant slave sp
505
506                     if      NmbSlvs GE 2
507                 Slave1:
508                     ds      52      ; processor descriptor
509                     ds      450     ; stack area for reentrant slave sp
510                     endif
511
512                     if      NmbSlvs GE 3
513                 Slave2:
514                     ds      52      ; processor descriptor
515                     ds      450     ; stack area for reentrant slave sp
516                     endif
```

```
517
518                     if      NmbSlvs GE 4
519             Slave3:
520                     ds      52      ; processor descriptor
521                     ds      450     ; stack area for reentrant slave sp
522                     endif
523
524                     if      mail
525             ;
526             ; Interprocessor Electronic Mail Data Segment
527             ;
528             MailBox:
529 01CC                 ds      2               ; link
530 01CE 4D61696C42     db      'MailBoxQ'      ; name
531 01D6 0200           dw      2               ; msglen
532 01D8 0100           dw      1               ; nmbmsgs
533 01DA                 ds      2               ; dqph
534 01DC                 ds      2               ; nqph
535 01DE                 ds      2               ; msgin
536 01E0                 ds      2               ; msgout
537 01E2                 ds      2               ; msgcnt
538 01E4                 ds      2               ; buffer
539
540 01E6        fcbtable:       ds      40*nmbfcbs*nmbSlvs
541
542             MailBoxUQCB:
543 0466 CC01           dw      MailBox         ; pointer
544 0468 6A04           dw      MailBoxAdr      ; msgadr
545
546             MailBoxAdr:
547 046A 6C04           dw      MailBoxes       ; Mail Box Data Structure Adr
548
549 0004 =      nmb$mail        equ     4       ; # pieces mail max per slave
550 0080 =      size$mail       equ     128     ; max size per piece of mail
551                                             ;  includes two byte overhead of
552                                             ;  source ID and size
553
554 0200 =      mail$buffer$size equ    nmb$mail*size$mail
555
556             Mail$boxes:
557 046C 04             db      nmb$mail
558 046D 80             db      size$mail
559
560             MstrMail:
561 046E 00             db      0       ; Master Mail In Ptr
562 046F 00             db      0       ; Master Mail Out Ptr
563 0470 00             db      0       ; Master Mail Cnt
564 0471 7804           dw      MstrMailBuffer  ; Master Mail Buffer Adr
565
```

```
566                     Slv0Mail:
567   0473 00                   db      0       ; Slave #0 Mail In Ptr
568   0474 00                   db      0       ; Slave #0 Mail Out Ptr
569   0475 00                   db      0       ; Slave #0 Mail Cnt
570   0476 7806                 dw      Slv0MailBuffer  ; Slave #0 Mail Buffer Adr
571
572                             if      NmbSlvs GE 2
573                     Slv1Mail:
574                             db      0       ; Slave #1 Mail In Ptr
575                             db      0       ; Slave #1 Mail Out Ptr
576                             db      0       ; Slave #1 Mail Cnt
577                             dw      Slv1MailBuffer  ; Slave #1 Mail Buffer Adr
578                             endif
579
580                             if      NmbSlvs GE 3
581                     Slv2Mail:
582                             db      0       ; Slave #2 Mail In Ptr
583                             db      0       ; Slave #2 Mail Out Ptr
584                             db      0       ; Slave #2 Mail Cnt
585                             dw      Slv2MailBuffer  ; Slave #2 Mail Buffer Adr
586                             endif
587
588                             if      NmbSlvs GE 4
589                     Slv3Mail:
590                             db      0       ; Slave #3 Mail In Ptr
591                             db      0       ; Slave #3 Mail Out Ptr
592                             db      0       ; Slave #3 Mail Cnt
593                             dw      Slv3MailBuffer  ; Slave #3 Mail Buffer Adr
594                             endif
595
596                     MstrMailBuffer:
597   0478                      ds      mail$buffer$size
598
599                     Slv0MailBuffer:
600   0678                      ds      mail$buffer$size
601
602                             if      NmbSlvs GE 2
603                     Slv1MailBuffer:
604                             ds      mail$buffer$size
605                             endif
606
607                             if      NmbSlvs GE 3
608                     Slv2MailBuffer:
609                             ds      mail$buffer$size
610                             endif
611
612                             if      NmbSlvs GE 4
613                     Slv3MailBuffer:
614                             ds      mail$buffer$size
615                             endif
616                             endif   ; matches if mail
```

```
                ;       Local Data Segment
                ;
                BinaryASCII:
087B FF                 db      0ffh    ; Slave #0: 0=7 bit ASCII, FF=8 bit binary
087C FF                 db      0ffh    ;       #1
087D FF                 db      0ffh    ;       #2
087E FF                 db      0ffh    ;       #3

                Networkstatus:
087F 00                 db      0       ; Slave #0 network status byte
0880 00                 db      0       ;       #1
0881 00                 db      0       ;       #2
0882 00                 db      0       ;       #3

0883 0000       conin:  dw      $-$     ; save area for XIOS routine address

000A =          max$retries     equ     10      ; maximum send message retries
                ;
                ;       The following tables are for use in the ALTOS i/o routines.
                ;       Note that this program MUST be used with an XIOS which allows
                ;       using the second printer port as a console port - Accessed as console
                ;       #4
                ;
002B =          Console4$status equ     02bh
002F =          Console3$status equ     02fh
002D =          Console2$status equ     02dh
0029 =          Printer2$status equ     029h    ; ALSO CONSOLE #4

                        if      z80
                ;
                ;       ENTRIES IN THE FOLLOWING TWO TABLES MUST MATCH !!!!
                ;
                status$ports:
                        db      Console4$status ; Console 4 (Slave 0) status port
                        db      Console3$status ; Console 3 (Slave 1) status port
                        db      Console2$status ; Console 2 (Slave 2) status port
                        db      Printer2$status ; Printer 2 (Slave 3) status port
                        endif

                chariotbl:                      ; Relationship between slaves and consoles
0884 03                 db      3
0885 02                 db      2
0886 01                 db      1
0887 04                 db      4
                ;       Network Status Byte Equates
                ;
0080 =          ntwrktxrdy      equ     10000000b       ; ntwrk I/F ready to send msg
0010 =          active          equ     00010000b       ; slave logged into network
0008 =          msgerr          equ     00001000b       ; error in received message
0004 =          ntwrk           equ     00000100b       ; network alive
0002 =          msgovr          equ     00000010b       ; message overrun
0001 =          ntwrkrxrdy      equ     00000001b       ; ntwrk I/F has rcvd msg
```

```
670
671                     ;       BDOS and XDOS Equates
672                     ;
673   0085 =            flagset equ     133     ; flag set
674   0086 =            makeq   equ     134     ; make queue
675   0089 =            readq   equ     137     ; read queue
676   008B =            writeq  equ     139     ; write queue
677   008D =            delay   equ     141     ; delay
678   008E =            dsptch  equ     142     ; dispatch
679   0090 =            createp equ     144     ; create process
680   009A =            sydatad equ     154     ; system data page address
681   0083 =            poll    equ     083h    ; Poll device
682
683                     ;       General Equates
684                     ;
685   0001 =            SOH     equ     01h     ; Start of Header
686   0002 =            STX     equ     02h     ; Start of Data
687   0003 =            ETX     equ     03h     ; End of Data
688   0004 =            EOT     equ     04h     ; End of Transmission
689   0005 =            ENQ     equ     05h     ; Enquire
690   0006 =            ACK     equ     06h     ; Acknowledge
691   000A =            LF      equ     0ah     ; Line Feed
692   000D =            CR      equ     0dh     ; Carriage Return
693   0015 =            NAK     equ     15h     ; Negative Acknowledge
694
695   0010 =            printer2        equ     10h     ; special poll device number for second
696                                                       ; printer port
697
698                     ;       Utility Procedures
699                     ;
700                     bdos:
701   0886 2A0000               lhld    bdosadr
702   0889 E9                   pchl
703
704                     Nibout:                 ; A = nibble to be transmitted in ASCII
705   088A FE0A                 cpi     10
706   088C D29508               jnc     nibatof ; jump if A-F
707   088F C630                 adi     '0'
708   0891 4F                   mov     c,a
709   0892 C39E08               jmp     Charout
710                     nibatof:
711   0895 C637                 adi     'A'-10
712   0897 4F                   mov     c,a
713   0898 C39E08               jmp     Charout
714
715                     PreCharout:
716   089B 7A                   mov     a,d
717   089C 81                   add     c
718   089D 57                   mov     d,a     ; update the checksum
719
```

```
                        if      z80
                char$out:
                ;
                ;       Character output routine for network i/o
                ;       using the ALTOS SIO ports
                ;
                ;       Entry: C register contains 8 bit value to transmit
                ;       Entry : Slave number in register b

                        push    h
                        push    d
                        push    b
                        mov     d, c            ; save the character
                        lxi     h, status$ports
                        mov     c, b
                        mvi     b, 0            ; set (BC) = (b)
                        dad     b
                        mov     c,m
                        ; Now C contains the address of the correct status port
                outputloop:
                        mvi     a, 10h
                ;       out     (c),a
                        db      0edh,79h
                ;       in      a,(c)
                        db      0edh,78h
                        ani     04h             ; wait for TXready
                        jz      outputloop

                        ; In the Altos system, data registers are one below status registers...
                        dcr     c
                ;       out     (c),d
                        db      0edh,51h
                        pop     b
                        pop     d
                        pop     h
                        ret

                        else

                char$out:
                ;       Character output routine for network I/O
                ;       using ALTOS SIO ports
                ;
                ;       Entry:  C = character to transmit
                ;               B = slave id byte

089E E5                 push    h
089F D5                 push    d
08A0 C5                 push    b
```

```
770  08A1 11AF08          lxi   d,out0              ; dispatch address =
771  08A4 68              mov   l,b                 ;   out0 + slaveid*16
772  08A5 2600            mvi   h,0
773  08A7 29              dad   h
774  08A8 29              dad   h
775  08A9 29              dad   h
776  08AA 29              dad   h
777  08AB 19              dad   d
778  08AC 3E10            mvi   a,10h               ;load 'get transmit status' value
779  08AE E9              pchl                      ;dispatch
780
781                 out0:
782  08AF D32B            out   Console4$status     ;wait for TXready status
783  08B1 DB2B            in    Console4$status
784  08B3 E604            ani   4
785  08B5 CAAF08          jz    out0
786
787  08B8 79              mov   a,c
788  08B9 D32A            out   Console4$status-1   ;write the character
789  08BB C1              pop   b
790  08BC D1              pop   d
791  08BD E1              pop   h
792  08BE C9              ret
793
794  08BF D32F      out1: out   Console3$status
795  08C1 DB2F            in    Console3$status
796  08C3 E604            ani   4
797  08C5 CABF08          jz    out1
798
799  08C8 79              mov   a,c
800  08C9 D32E            out   Console3$status-1
801  08CB C1              pop   b
802  08CC D1              pop   d
803  08CD E1              pop   h
804  08CE C9              ret
805
806  08CF D32D      out2: out   Console2$status
807  08D1 DB2D            in    Console2$status
808  08D3 E604            ani   4
809  08D5 CACF08          jz    out2
810
811  08D8 79              mov   a,c
812  08D9 D32C            out   Console2$status-1
813  08DB C1              pop   b
814  08DC D1              pop   d
815  08DD E1              pop   h
816  08DE C9              ret
817
818  08DF D329      out3: out   Printer2$status
819  08E1 DB29            in    Printer2$status
820  08E3 E604            ani   4
821  08E5 CADF08          jz    out3
```

```
822
823  08E8 79            mov     a,c
824  08E9 D328          out     Printer2$status-1
825  08EB C1            pop     b
826  08EC D1            pop     d
827  08ED E1            pop     h
828  08EE C9            ret
829
830                     endif
831
832
833             ;
834
835             Nibin:          ; return nibble in A register
836  08EF CD2609        call    Charin
837  08F2 D8            rc
838  08F3 E67F          ani     07fh
839  08F5 D630          sui     '0'
840  08F7 FE0A          cpi     10
841  08F9 DA0F09        jc      Nibin$return ; must be 0-9
842  08FC C6F9          adi     ('0'-'A'+10) and 0ffh
843  08FE FE10          cpi     16
844  0900 DA0F09        jc      Nibin$return ; must be 10-15
845  0903 3A7C08        lda     networkstatus
846  0906 F608          ori     msgerr
847  0908 327C08        sta     networkstatus
848  090B 3E00          mvi     a,0
849  090D 37            stc
850  090E C9            ret
851             Nibin$return:
852  090F B7            ora     a
853  0910 C9            ret
854
855             xChar$in:
856             ;
857  0911 E5            push    h
858  0912 C5            push    b
859  0913 212309        lxi     h, Charin$return
860  0916 E5            push    h
861  0917 48            mov     c,b
862  0918 0600          mvi     b,0
863  091A 218208        lxi     h, chariotbl
864  091D 09            dad     b
865  091E 56            mov     d, m    ; Get the console number
866  091F 2A8008        lhld    conin
867  0922 E9            pchl            ; vector off
868             Charin$return:
869  0923 C1            pop     b
870  0924 E1            pop     h
871  0925 C9            ret
872
873
```

```
874                     if      z80
875                 char$in:
876                 ;
877                 ;       Character input routine for network i/o
878                 ;       using the ALTOS SIO ports at 125k baud
879                 ;
880                 ;
881                 ;       Entry : Slave number in register b
882                 ;       Exit  : Character in register a
883                 ;
884                         push    h
885                         push    b
886                         lxi     h, status$ports
887                         mov     c, b
888                         mvi     b, 0            ; set (BC) = (b)
889                         dad     b
890                         mov     c,m
891                         ; Now C contains the address of the correct status port
892                         mvi     l, 80
893                 inputloop1:
894                         dcr     l
895                         jz      retout
896                 ;       in      a,(c)
897                         db      0edh,78h
898                         ani     01h             ; wait for RXready
899                         jz      inputloop1
900
901                         ; In the Altos system, data registers are one below status registers...
902                         dcr     c
903                 ;       in      a,(c)   ; Get the character
904                         db      0edh,78h
905                         pop     b
906                         pop     h
907                         ret
908
909                 retout:
910                         stc             ;set carry => error flag
911                         pop     b
912                         pop     h
913                         ret
914
915                         else
916
917                 char$in:
918                 ;       Character input routine for network I/O
919                 ;       using ALTOS SIO ports
920                 ;
921                 ;       Entry:  B = Slave ID
922                 ;       Exit:   A = character input
923
924  0926 E5                push    h
925  0927 D5                push    d
```

```
926  0928 C5            push    b
927  0929 113A09        lxi     d,in0               ; HL = in0 + 17*slaveid
928  092C 68            mov     l,b
929  092D 2600          mvi     h,0
930  092F EB            xchg
931  0930 19            dad     d
932  0931 EB            xchg
933  0932 29            dad     h
934  0933 29            dad     h
935  0934 29            dad     h
936  0935 29            dad     h
937  0936 19            dad     d
938
939  0937 0E50          mvi     c,80                ; load status retry count
940  0939 E9            pchl                        ; dispatch
941
942                in0:
943  093A 0D            dcr     c
944  093B CA7E09        jz      retout              ; error return if retry timeout
945
946  093E DB2B          in      Console4$status     ; wait for RXready
947  0940 E601          ani     1
948  0942 CA3A09        jz      in0
949
950  0945 DB2A          in      Console4$status-1   ; get the character
951  0947 C1            pop     b
952  0948 D1            pop     d
953  0949 E1            pop     h
954  094A C9            ret
955
956                in1:
957  094B 0D            dcr     c
958  094C CA7E09        jz      retout              ; error return if retry timeout
959
960  094F DB2F          in      Console3$status     ; wait for RXready
961  0951 E601          ani     1
962  0953 CA4B09        jz      in1
963
964  0956 DB2E          in      Console3$status-1   ; get the character
965  0958 C1            pop     b
966  0959 D1            pop     d
967  095A E1            pop     h
968  095B C9            ret
969
970                in2:
971  095C 0D            dcr     c
972  095D CA7E09        jz      retout              ; error return if retry timeout
973
974  0960 DB2D          in      Console2$status     ; wait for RXready
975  0962 E601          ani     1
976  0964 CA5C09        jz      in2
977
```

```
 978  0967 DB2C          in      Console2$status-1       ; get the character
 979  0969 C1            pop     b
 980  096A D1            pop     d
 981  096B E1            pop     h
 982  096C C9            ret
 983                in3:
 984  096D 0D            dcr     c
 985  096E CA7E09        jz      retout                  ; error return if retry timeout
 986
 987  0971 DB29          in      Printer2$status         ; wait for RXready
 988  0973 E601          ani     1
 989  0975 CA6D09        jz      in3
 990
 991  0978 DB28          in      Printer2$status-1       ; get the character
 992  097A C1            pop     b
 993  097B D1            pop     d
 994  097C E1            pop     h
 995  097D C9            ret
 996
 997                retout:                             ; error return (carry=1)
 998  097E 37            stc
 999  097F C1            pop     b
1000  0980 D1            pop     d
1001  0981 E1            pop     h
1002  0982 C9            ret
1003
1004                    endif
1005
1006
1007                Netout: ; C = byte to be transmitted
1008  0983 7A            mov     a,d
1009  0984 81            add     c
1010  0985 57            mov     d,a
1011  0986 3A7808        lda     BinaryASCII
1012  0989 B7            ora     a
1013  098A C29E08        jnz     Charout ; transmit byte in Binary mode
1014  098D 79            mov     a,c
1015  098E F5            push    psw
1016  098F 1F            rar
1017  0990 1F            rar
1018  0991 1F            rar
1019  0992 1F            rar
1020  0993 E60F          ani     0FH     ; mask HO nibble to LO nibble
1021  0995 CD8A08        call    Nibout
1022  0998 F1            pop     psw
1023  0999 E60F          ani     0FH
1024  099B C38A08        jmp     Nibout
1025
```

```
1026                    Netin:          ; byte returned in A register
1027                                    ; D  = checksum accumulator
1028  099E 3A7808               lda     BinaryASCII
1029  09A1 B7                   ora     a
1030  09A2 CAAC09               jz      ASCIIin
1031  09A5 CD2609               call    charin  ;receive byte in Binary mode
1032  09A8 D8                   rc
1033  09A9 C3BC09               jmp     chksin
1034                    ASCIIin:
1035  09AC CDEF08               call    Nibin
1036  09AF D8                   rc
1037  09B0 87                   add     a
1038  09B1 87                   add     a
1039  09B2 87                   add     a
1040  09B3 87                   add     a
1041  09B4 F5                   push    psw
1042  09B5 CDEF08               call    Nibin
1043  09B8 D8                   rc
1044  09B9 E3                   xthl
1045  09BA B4                   ora     h
1046  09BB E1                   pop     h
1047                    chksin:
1048  09BC B7                   ora     a
1049  09BD F5                   push    psw
1050  09BE 82                   add     d       ; add & update checksum accum.
1051  09BF 57                   mov     d,a
1052  09C0 F1                   pop     psw
1053  09C1 C9                   ret
1054
1055                    Msgin:          ; HL = destination address
1056                                    ; E  = # bytes to input
1057  09C2 CD9E09               call    Netin
1058  09C5 D8                   rc
1059  09C6 77                   mov     m,a
1060  09C7 23                   inx     h
1061  09C8 1D                   dcr     e
1062  09C9 C2C209               jnz     Msgin
1063  09CC C9                   ret
1064
1065                    Msgout: ; HL = source address
1066                            ; E  = # bytes to output
1067                            ; D  = checksum
1068                            ; C  = preamble character
1069  09CD 1600                 mvi     d,0
1070  09CF CD9B08               call    PreCharout
1071                    Msgoutloop:
1072  09D2 4E                   mov     c,m
1073  09D3 23                   inx     h
1074  09D4 CD8309               call    Netout
1075  09D7 1D                   dcr     e
1076  09D8 C2D209               jnz     Msgoutloop
1077  09DB C9                   ret
```

```
1078
1079                    ;       Network Initialization
1080                    nwinit:
1081                    ;       device initialization, as required
1082                    ;
1083                    ;
1084    09DC 3E47               mvi     a,047h          ;sets up CTC for baud rate of 125K
1085    09DE D331               out     031h
1086    09E0 D330               out     030h
1087    09E2 D332               out     032h
1088    09E4 3E01               mvi     a,1             ;count of one => max speed
1089    09E6 D331               out     031h
1090    09E8 D330               out     030h            ; slave #1 ctc channel
1091    09EA D332               out     032h
1092                    ;
1093                    ;       Find address of XIOS console output routine
1094                    ;
1095    09EC 2A0100             lhld    0001h
1096    09EF 23                 inx     h
1097    09F0 5E                 mov     e, m
1098    09F1 23                 inx     h
1099    09F2 56                 mov     d, m
1100    09F3 210600             lxi     h, 0006h        ; Offset for conin routine
1101    09F6 19                 dad     d
1102    09F7 228008             shld    conin           ; save the address
1103    09FA AF                 xra     a               ;return code is 0=success
1104    09FB C9                 ret
1105
1106
1107                    ;       Network Status
1108                    nwstat:         ; C = Slave #
1109    09FC 0600               mvi     b,0
1110    09FE 217C08             lxi     h,networkstatus
1111    0A01 09                 dad     b
1112    0A02 7E                 mov     a,m
1113    0A03 47                 mov     b,a
1114    0A04 E6F5               ani     not (msgerr+msgovr)
1115    0A06 77                 mov     m,a
1116    0A07 78                 mov     a,b
1117    0A08 C9                 ret
1118
1119
1120                    ;       Return Configuration Table Address
1121                    cfgadr:
1122    0A09 21AB01             lxi     h,configtbl
1123    0A0C C9                 ret
1124
1125
1126                    ;       Send Message on Network
1127                    sndmsg:                 ; DE = message addr
1128                                            ;  C = Slave #
1129    0A0D 41                 mov     b,c
1130    0A0E 3E0A               mvi     a,max$retries   ; A = max$retries
```

```
1131                    send:
1132  0A10 F5                   push    psw
1133
1134                            if      mutexout
1135                            push    b
1136                            push    d
1137                            mvi     c,readq
1138                            lxi     d,UQCBMXSXmitq
1139                            call    bdos    ; obtain mutual exclusion message
1140                            pop     d
1141                            pop     b
1142                            endif
1143
1144  0A11 EB                   xchg
1145  0A12 E5                   push    h
1146  0A13 F3                   di
1147  0A14 0E05                 mvi     c,ENQ
1148  0A16 CD9E08               call    Charout ; send ENQ
1149  0A19 CD4D0A               call    getACK
1150  0A1C 1E05                 mvi     e,5
1151  0A1E 0E01                 mvi     c,SOH
1152  0A20 CDCD09               call    Msgout  ; send SOH FMT DID SID FNC SIZ
1153  0A23 AF                   xra     a
1154  0A24 92                   sub     d
1155  0A25 4F                   mov     c,a
1156  0A26 CD8309               call    Netout  ; send HCS (header checksum)
1157  0A29 CD4D0A               call    getACK
1158  0A2C 2B                   dcx     h
1159  0A2D 5E                   mov     e,m
1160  0A2E 23                   inx     h
1161  0A2F 1C                   inr     e
1162  0A30 0E02                 mvi     c,STX
1163  0A32 CDCD09               call    Msgout  ; send STX DB0 DB1 ...
1164  0A35 0E03                 mvi     c,ETX
1165  0A37 CD9B08               call    PreCharout      ; send ETX
1166  0A3A AF                   xra     a
1167  0A3B 92                   sub     d
1168  0A3C 4F                   mov     c,a
1169  0A3D CD8309               call    Netout  ; send CKS
1170  0A40 0E04                 mvi     c,EOT
1171  0A42 CD9B08               call    PreCharout      ; send EOT
1172  0A45 CD4D0A               call    getACK
1173  0A48 D1                   pop     d       ; discard message address
1174  0A49 F1                   pop     psw     ; discard retry counter
1175
1176                            if      mutexout
1177                            call    release$MX
1178                            endif
```

```
1179
1180  0A4A FB             ei
1181  0A4B AF             xra     a
1182  0A4C C9             ret             ; A = 0, successful send message
1183
1184                getACK:
1185  0A4D CD2609         call    Charin
1186  0A50 DA580A         jc      getACK$timeout
1187  0A53 E67F           ani     7fh
1188  0A55 D606           sui     ACK
1189  0A57 C8             rz
1190                getACK$timeout:
1191  0A58 D1             pop     d       ; discard return address
1192
1193                      if      mutexout
1194                      push    b
1195                      call    release$MX
1196                      pop     b
1197                      endif
1198
1199  0A59 D1             pop     d       ; DE = message address
1200  0A5A F1             pop     psw     ; A = retry count
1201  0A5B 3D             dcr     a
1202  0A5C C2100A         jnz     send
1203  0A5F 3D             dcr     a       ; A = 0ffh
1204  0A60 C9             ret             ; failed to send message
1205
1206                      if      mutexin or mutexout
1207               release$MX:
1208                      ; send back slave transmit MX message
1209                      mvi     c,writeq
1210                      lxi     d,UQCBMXSXmitq
1211                      jmp     bdos
1212                      endif
1213
1214               ;      Receive Message from Network
1215               rcvmsg:                 ; DE = message addr
1216                                       ;  C = Slave #
1217  0A61 41             mov     b,c
1218               receive:
1219  0A62 EB             xchg
1220  0A63 E5             push    h
1221  0A64 CD6C0A         call    get$ENQ
1222                      ; a return to this point indicates an error
1223
1224               receive$retry:
1225  0A67 FB             ei              ;enable other processes
1226
1227                      if      mutexin
1228                      push    b
1229                      call    release$MX
1230                      pop     b
1231                      endif
```

```
1232
1233  0A68 D1           pop     d
1234  0A69 C3620A       jmp     receive
1235
1236                get$ENQ:
1237  0A6C CD1109       call    xCharin
1238  0A6F DA6C0A       jc      get$ENQ
1239  0A72 E67F         ani     7fh
1240  0A74 FE05         cpi     ENQ     ; Start of Header ?
1241  0A76 C26C0A       jnz     get$ENQ
1242
1243                    if      mutexin
1244                    ;get the slave xmit MX message before allowing slv to xmit
1245                    push    b
1246                    push    h
1247                    mvi     c,readq
1248                    lxi     d,UQCBMXSXmitq
1249                    call    bdos
1250                    pop     h
1251                    pop     b
1252                    endif
1253
1254  0A79 0E06         mvi     c,ACK
1255  0A7B F3           di              ;slave in gear now serve only him
1256
1257  0A7C CD9E08       call    charout ; send ACK to slave, allows it to xmit
1258  0A7F CD2609       call    Charin
1259  0A82 D8           rc
1260  0A83 E67F         ani     7fh
1261  0A85 FE01         cpi     SOH
1262  0A87 C0           rnz
1263  0A88 57           mov     d,a     ; initialize the HCS
1264  0A89 1E05         mvi     e,5
1265  0A8B CDC209       call    Msgin
1266  0A8E D49E09       cnc     Netin
1267  0A91 D8           rc
1268  0A92 7A           mov     a,d
1269  0A93 B7           ora     a
1270  0A94 C2C10A       jnz     sendNAK ; jmp & send NAK if HCS <> 0
1271  0A97 0E06         mvi     c,ACK
1272  0A99 CD9E08       call    Charout
1273  0A9C CD2609       call    Charin
1274  0A9F D8           rc
1275  0AA0 E67F         ani     7fh
1276  0AA2 FE02         cpi     STX
1277  0AA4 C0           rnz
1278  0AA5 57           mov     d,a     ; initialize the CKS
1279  0AA6 2B           dcx     h
1280  0AA7 5E           mov     e,m
1281  0AA8 23           inx     h
1282  0AA9 1C           inr     e
1283  0AAA CDC209       call    msgin
```

```
1284  0AAD D42609          cnc     Charin
1285  0AB0 D8              rc
1286  0AB1 E67F            ani     7fh
1287  0AB3 FE03            cpi     ETX
1288  0AB5 C0              rnz
1289  0AB6 82              add     d
1290  0AB7 57              mov     d,a
1291  0AB8 CD9E09          call    Netin   ; get Checksum byte
1292  0ABB D8              rc
1293  0ABC 7A              mov     a,d
1294  0ABD B7              ora     a       ; should be zero
1295  0ABE CAC60A          jz      sendACK ; jump if checksum OK
1296                    sendNAK:
1297  0AC1 0E15            mvi     c,NAK
1298  0AC3 C39E08          jmp     Charout ; send NAK and return to receive$retry
1299                    sendACK:
1300  0AC6 CD2609          call    Charin  ; get EOT
1301  0AC9 D8              rc
1302  0ACA E67F            ani     7fh
1303  0ACC FE04            cpi     EOT
1304  0ACE C0              rnz
1305  0ACF 0E06            mvi     c,ACK
1306  0AD1 CD9E08          call    Charout ; send ACK if checksum ok
1307  0AD4 D1              pop     d       ; discard return address
1308  0AD5 D1              pop     d       ; discard message address
1309  0AD6 FB              ei              ; now allow multiprocessing
1310
1311                       if      mutexin
1312                       call    release$MX
1313                       endif
1314
1315  0AD7 AF              xra     a
1316  0AD8 C9              ret
1317
1318
1319
1320                    restore:
1321  0AD9 F3              di
1322  0ADA E1              pop     h
1323  0ADB 22ED0A          shld    rtnadr
1324  0ADE E1              pop     h
1325  0ADF D1              pop     d
1326  0AE0 C1              pop     b
1327  0AE1 F1              pop     psw
1328  0AE2 F5              push    psw
1329  0AE3 C5              push    b
1330  0AE4 D5              push    d
1331  0AE5 E5              push    h
1332  0AE6 E5              push    h
1333  0AE7 2AED0A          lhld    rtnadr
1334  0AEA E3              xthl
1335  0AEB FB              ei
1336  0AEC C9              ret
1337  0AED              rtnadr: ds      2
```

```
1338
1339                    if      WtchDg
1340            ;  WatchDog Timer Process
1341            ;
1342            WatchDog:
1343                    mvi     c,Delay
1344                    lxi     d,60    ; dly for 1 second
1345                    call    bdos
1346                    lhld    WatchDogTime
1347                    inx     h
1348                    shld    WatchDogTime
1349                    lxi     h,WatchDogTable-5
1350                    mvi     c,NmbSlvs
1351            WatchDogLoop:
1352                    lxi     d,0005h
1353                    dad     d
1354                    mov     a,m
1355                    ora     a
1356                    jz      WatchDogDec
1357                    inx     h
1358                    ana     m
1359                    dcx     h
1360                    jnz     WatchDogDec     ; waiting & timeout set
1361                    push    h               ; save HL -> WDT.waiting
1362                    inx     h
1363                    inx     h
1364                    di
1365                    mov     e,m
1366                    inx     h
1367                    mov     d,m
1368                    ei
1369                    lhld    WatchDogTime
1370                    mov     a,l
1371                    sub     e
1372                    mov     l,a
1373                    mov     a,h
1374                    sbb     d
1375                    mov     h,a
1376                    mvi     a,10    ; # seconds since started Charinn
1377                    sub     l
1378                    mvi     a,0
1379                    sbb     h
1380                    pop     h
1381                    jnc     WatchDogDec
1382                    push    h
1383                    inx     h
1384                    mvi     m,0ffh  ; WDT.timeout = 0ffh
1385                    inx     h
1386                    inx     h
1387                    inx     h
1388                    push    b
1389                    mov     e,m     ; E = Flag #
```

```
1390                            mvi     c,Flagset
1391                            call    bdos
1392                            pop     b
1393                            pop     h
1394
1395                    WatchDogDec:
1396                            dcr     c
1397                            jnz     WatchDogLoop
1398
1399                            jmp     WatchDog
1400                            endif
1401
1402                    ;
1403                    ;       Setup code for Network Interface Procedures
1404                    ;
1405                    Setup:
1406    0AEF F5                 push    psw
1407    0AF0 C5                 push    b
1408    0AF1 D5                 push    d
1409    0AF2 E5                 push    h
1410    0AF3 CDDC09             call    nwinit
1411
1412                            if      mutexin or mutexout
1413                            mvi     c,makeq
1414                            lxi     d,QCBMXSXmitq
1415                            call    bdos
1416                            mvi     c,writeq
1417                            lxi     d,UQCBMXSXmitq
1418                            call    bdos
1419                            endif
1420
1421                            if      mail
1422    0AF6 0E86               mvi     c,makeq
1423    0AF8 11CC01             lxi     d,MailBox
1424    0AFB CD8608             call    bdos
1425    0AFE 0E8B               mvi     c,writeq
1426    0B00 116604             lxi     d,MailBoxUQCB
1427    0B03 CD8608             call    bdos
1428                            endif
1429
1430                            if      WtchDg
1431                            lxi     d,WatchDogPD
1432                            mvi     c,createp
1433                            call    bdos
1434                            else
1435                            if      NmbSlvs GE 2
1436                            lxi     d,NtwrkIP1
1437                            mvi     c,createp
1438                            call    bdos
1439                            endif
1440                            endif
```

```
1441
1442  0B06 0E8E             mvi     c,dsptch
1443  0B08 CD8608           call    bdos
1444  0B0B 0E9A             mvi     c,sydatad
1445  0B0D CD8608           call    bdos
1446  0B10 110900           lxi     d,9
1447  0B13 19               dad     d
1448  0B14 11AB01           lxi     d,configtbl
1449  0B17 73               mov     m,e
1450  0B18 23               inx     h
1451  0B19 72               mov     m,d     ; sysdatpage(9&10) = co.configtbl
1452
1453                        if      modem
1454                        ; Initialize the modem
1455                        mvi     c,CR
1456                        mvi     b,slvmodem
1457                        call    Charout
1458                        mvi     c,'Z'
1459                        call    Charout
1460                        mvi     c,CR
1461                        call    Charout
1462                  WtSpace:
1463                        call    Charin
1464                        jc      SetupDone
1465                        ani     07fh
1466                        cpi     ' '
1467                        jnz     WtSpace
1468                        mvi     c,'A'
1469                        call    Charout
1470                  SetupDone:
1471                        endif
1472
1473  0B1A E1               pop     h
1474  0B1B D1               pop     d
1475  0B1C C1               pop     b
1476  0B1D F1               pop     psw
1477                  ;
1478                  ;  Network Interface Reentrant Procedure
1479                  ;
1480                  Init:
1481  0B1E F5               push    psw     ; A = network i/f console dev #
1482  0B1F C5               push    B       ; BC= buffer address
1483  0B20 D5               push    D       ; DE= UQCB ntwrk queue out
1484  0B21 E5               push    H       ; HL= UQCB ntwrk queue in
1485  0B22 5E               mov     e,m
1486  0B23 23               inx     h
1487  0B24 56               mov     d,m
1488  0B25 0E86             mvi     c,makeq
1489  0B27 CD8608           call    bdos    ; make the ntwrk queue in
1490  0B2A CDD90A           call    restore
1491  0B2D EB               xchg
1492  0B2E 5E               mov     e,m
```

```
1493  0B2F 23           inx   h
1494  0B30 56           mov   d,m
1495  0B31 0E86         mvi   c,makeq
1496  0B33 CD8608       call  bdos    ; make the ntwrk queue out
1497
1498              Loop:
1499  0B36 CDD90A       call  restore
1500  0B39 50           mov   d,b
1501  0B3A 59           mov   e,c
1502              ;
1503              ;
1504  0B3B 4F           mov   c,a
1505  0B3C CD610A       call  rcvmsg
1506
1507  0B3F CDD90A       call  restore
1508  0B42 EB           xchg
1509  0B43 0E8B         mvi   c,writeq
1510  0B45 CD8608       call  bdos
1511
1512  0B48 CDD90A       call  restore
1513  0B4B 0E89         mvi   c,readq
1514  0B4D CD8608       call  bdos
1515
1516  0B50 CDD90A       call  restore
1517  0B53 50           mov   d,b
1518  0B54 59           mov   e,c
1519              ;
1520  0B55 4F           mov   c,a
1521  0B56 CD0D0A       call  sndmsg
1522
1523  0B59 C3360B       jmp   Loop
1524
1525  0B5C              end
```

```
MAXRETRIES      000A  634‡ 1130
MODEM           0000   29‡ 1453
MSGERR          0008  666‡  846  1114
MSGIN           09C2 1055‡ 1062  1265  1283
MSGOUT          09CD 1065‡ 1152  1163
MSGOUTLOOP      09D2 1071‡ 1076
MSGOVR          0002  668‡ 1114
MSTRMAIL        046E  560‡
MSTRMAILBUFFER  0478  564   596‡
MUTEXIN         0000   35‡  465  1206  1227  1243  1311  1412
MUTEXOUT        0000   36‡  465  1134  1176  1193  1206  1412
NAK             0015  693‡ 1297
NETIN           099E 1026‡ 1057  1266  1291
NETOUT          0983 1007‡ 1074  1156  1169
NETWORKSTATUS   087C  626‡  845   847  1110
NIBATOF         0895  706   710‡
NIBIN           08EF  835‡ 1035  1042
NIBINRETURN     090F  841   844   851‡
NIBOUT          088A  704‡ 1021  1024
NMBFCBS         0010   38‡  497   540
NMBMAIL         0004  549‡  554   557
NMBSLVS         0001   41‡   58‡  148   150   237   239   326   414   489   506
                       512   518   540   572   580   588   602   607   612  1350
                      1435
NMSG            0001  500‡
NTWRK           0004  667‡
NTWRKIPO        0002   65‡
NTWRKISO        0036   43    69    93‡
NTWRKRXRDY      0001  669‡
NTWRKTXRDY      0080  664‡
NWINIT          09DC 1080‡ 1410
NWSTAT          09FC 1108‡
OUT0            08AF  770   781‡  785
OUT1            08BF  794‡  797
OUT2            08CF  806‡  809
OUT3            08DF  818‡  821
POLL            0083  681‡
PRECHAROUT      089B  715‡ 1070  1165  1171
PRINTER2        0010  695‡
PRINTER2STATUS  0029  644‡  654   818   819   824   987   991
QCBNTWRKQIO     0066  102‡  115
QCBNTWRKQDO     0086  120‡  133
RCVMSG          0A61 1215‡ 1505
READQ           0089  675‡ 1137  1247  1513
RECEIVE         0A62 1218‡ 1234
RECEIVERETRY    0A67 1224‡
RESTORE         0AD9 1320‡ 1490  1499  1507  1512  1516
RETOUT          097E  895   909‡  944   958   972   985   997‡
RTNADR          0AED 1323  1333  1337‡
SEND            0A10 1131‡ 1202
SENDACK         0AC6 1295  1299‡
SENDNAK         0AC1 1270  1296‡
SETUP           0AEF  100  1405‡
```

SIZEMAIL	0080	550#	554	558		
SLVOMAIL	0473	566#				
SLVOMAILBUFFER	0678	570	599#			
SNDMSG	0A0D	1127#	1521			
SOH	0001	685#	1151	1261		
STX	0002	686#	1162	1276		
SYDATAD	009A	680#	1444			
TRUE	FFFF	24#	31			
UQCBNTWRKQI0	0080	47	87	114#		
UQCBNTWRKQO0	00A0	48	88	132#		
WRITEQ	008B	676#	1209	1416	1425	1509
WTCHDG	0000	33#	410	1339	1430	
XCHARIN	0911	855#	1237			
Z80	0000	26#	646	720	874	

Index